DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Design and Implementation of a Conversational Health Question Answering System

Jonas Lossin

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor’s Thesis in Informatics

Design and Implementation of a Conversational Health Question Answering System

Entwurf und Implementierung eines konversationsfähigen Systems zur Beantwortung von Gesundheitsfragen

Author:	Jonas Lossin
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M.Sc. Phillip Schneider
Submission Date:	15.02.2024

Acknowledgments

I would like to express my gratitude to those who have contributed to the completion of this thesis. First, a special thank you to M.Sc. Phillip Schneider, the advisor of my thesis. Not only did he provide valuable guidance along the way but also offered constructive feedback when I faced challenges or uncertainties.

Furthermore, I would like to thank my supervisor, Prof. Dr. Florian Matthes, for giving me the opportunity to conduct my thesis under the Chair of Software Engineering for Business Information Systems and contributing to current research.

Abstract

Conversational agents offer a more accessible method of retrieving information than traditional search engines. In contrast to these more impersonal tools, such as Google or Wikipedia, conversational agents offer a conversation-like interaction which enhances the user experience by asking follow-up questions and creating a sense of dialogue. This interactive approach is particularly valuable in healthcare, where users often include older adults less familiar with technology and individuals facing loneliness.

This thesis presents the development of a voice-based conversational agent that answers consumer health questions. At its core, this agent uses a similarity-based approach, employing a dataset of question-answer pairs as its knowledge base. As such medical question-answering datasets barely exist in the literature, this work addresses the gap by developing a high-quality dataset. For this purpose, a pipeline consisting of several steps is created, which automatically generates question-answer pairs with the help of a large language model.

Moreover, the agent contains a recommendation system, suggesting new questions to the user and therefore enhancing the conversation-like feeling. The efficacy as well as the user experience offered by the agent were assessed with two different strategies. The results of the evaluation confirm that the presented approach improves accessibility of health information and maintains a pleasant conversational experience.

Keywords: Conversational agent, question answering, consumer health questions, natural language processing, large language model, question-answer dataset

Kurzfassung

Gesprächsagenten bieten eine zugänglichere Methode zum Abrufen von Informationen als herkömmliche Suchmaschinen. Im Gegensatz zu diesen eher unpersönlichen Anwendungen wie Google oder Wikipedia bieten Conversational Agents eine konversationsähnliche Interaktion, die das Nutzererlebnis durch Folgefragen und die Vermittlung eines Dialoggefühls verbessert. Dieser interaktive Ansatz ist besonders wertvoll im Pflegebereich, wo zu den Nutzern häufig ältere Menschen gehören, die mit Technologie weniger vertraut sind und mit Einsamkeit zu kämpfen haben.

In dieser Arbeit wird die Entwicklung eines sprachbasierten Gesprächsagenten vorgestellt, der Gesundheitsfragen von Verbrauchern beantwortet. Im Kern verwendet dieser Agent einen ähnlichkeitsbasierten Ansatz, der einen Datensatz von Frage-Antwort-Paaren als Wissensbasis nutzt. Da solche Datensätze zur Beantwortung medizinischer Fragen in der Literatur kaum existieren, schließt diese Arbeit die Forschungslücke durch die Entwicklung eines hochwertigen Datensatzes. Dazu wird unter anderem eines großen Sprachmodells zur automatichen Generierung von Frage-Antwort-Paaren verwendet.

Darüber hinaus enthält der Agent ein Empfehlungssystem, das dem Nutzer neue Fragen vorschlägt und so das Gesprächsgefühl verstärkt. Die Wirksamkeit und das Benutzererlebnis des Agenten wurden mit zwei verschiedenen Strategien bewertet, wobei sich bestätigte, dass der vorgestellte Ansatz die Zugänglichkeit von Gesundheitsinformationen verbessert und ein angenehmes Gesprächserlebnis bietet.

Schlüsselwörter: Gesprächsagent, Beantwortung von Fragen, Gesundheitsfragen von Verbrauchern, Natural Language Processing, großes Sprachmodell, Frage-Antwort Datensatz

Contents

1. Introduction

1.1. Motivation

In today's age there is a vast amount of data available through the internet. Even though technically everyone can access the data, not everyone is consulting the internet to answer their questions. This might be due to the challenge of finding the appropriate information when using a search engine to answer a question. Although there is most likely an answer to one's question, it remains a challenge to find that piece of information. Often it requires reading through many articles to find a satisfying answer. Additionally, some people are less familiar with technology and therefore do not even attempt to enquire any search engine.

Conversational question-answering (QA) systems, also referred to as conversational agent (CA), try to overcome this challenge by providing answers in natural language. These systems can be categorized into two different types which are designed for different questions: Firstly, there are open-domain systems which are meant for questions from a wide range of domains. Secondly, there are restricted-domain systems which are specialized on one domain and aim to give more detailed answers [1].

In the scope of this thesis, we investigate approaches on restricted domain CAs, which focus on answering consumer health questions (CHQs). Although this limits the number of domains to only the medical domain, creating an effective system is still a significant challenge [2]. Besides the lack of high-quality datasets for QA in the consumer health domain, this is due to the gap between consumer language and medical jargon. Therefore, this thesis aims to provide more insights on how to implement and evaluate a CA that can effectively answer CHQs.

The development of this thesis is in cooperation with an ongoing research project, called *ALPHA-KI*[1]. The ALPHA-KI project focuses on the design of *ALPHA*, a smartwatch mainly used by patients in healthcare that only consists of a voice-based interface. The agent created in the scope of this thesis will be deployed as a new feature on ALPHA, designed to educate patients on health topics.

Since the smartwatch is being used in healthcare, the user group consists of mainly elderly people which are less familiar with technology. This group of users would also

[1]ALPHA-KI: `https://wwwmatthes.in.tum.de/pages/uysghltybqze/AI-Based-Digital-Health-Assistant-ALPHA-KI`

rarely consult any search engines when having health-related questions. Thus, the agent presented aims to improve the accessibility to health-related information especially for elder people.

Additionally, the smartwatch is designed for German speaking users and therefore the agent discussed throughout this thesis will also operate in German.

The agent mainly builds on a similarity-based approach to answer CHQs. When a user asks a question, the system searches through numerous stored questions to find the closest match. Each question is linked to a valid answer, which is then used to respond to the user. Compared to systems that heavily rely on documents written in professional language, the questions in our approach are written in everyday language which bridges the gap between medical jargon and consumer language. Thus, we aim to explore the use of such a similarity-based system to answer CHQs.

In order to achieve satisfying results with a similarity-based approach, a significant amount of QA pairs is required: The more unique questions we have stored, the more likely our agent can match an incoming question. Therefore, in this bachelor's thesis, we explore how to construct a dataset of QA pairs by presenting an automated pipeline and discussing the results. This contributes to the existing lack of high quality medical QA datasets available in literature [3].

1.2. Research

Based on the described motivation, we defined four research questions building the guideline of this thesis:

RQ1: Which existing approaches are there in building conversational question answering agents in the context of health-related topics?

The first question aims to give an overview of existing solutions in conversational QA, specifically in the medical domain. It covers two main areas: existing approaches for the automatic creation of QA pairs using large language models (LLMs) and the development of a CA which answers CHQs. These two areas represent the main pillars of this thesis.

RQ2: How to construct a dataset with question-answer pairs?

The second question addresses the first pillar of this thesis which is the automatic generation of QA pairs. To answer this question, we will explore how we can collect appropriate data, how to instruct an LLM to generate high quality QA pairs and how to prepare this dataset for further use by a conversational agent.

RQ3: How to develop a pipeline for conversational question answering using the scraped dataset?

Question three then focuses on the second pillar of this thesis which is the development of a CA. This question relies on the results from RQ2, considering that our agent will perform better the more unique questions we have generated in the previous step. In the scope of RQ3, we then explain the architecture of our agent, containing a pipeline of multiple steps in order to answer an incoming question.

RQ4: Which evaluation methods can be used to assess the performance and effectiveness of the developed system?

The final question aims to provide suitable evaluation methods for a CA. This is achieved by presenting an automatic evaluation approach as well as analyzing feedback given by a group of test users. Moreover, we discuss the results from both evaluation strategies.

1.3. Thesis Outline

The structure of this thesis is as follows: The motivation, research topics, and thesis structure are introduced in Chapter 1. Chapter 2 explains the foundations and theoretical background on the topics handled by this thesis. In Chapter 3, relevant research on automatic QA pair generation and CAs for CHQs is reviewed. The methods used to answer each research question are described in Chapter 4. The process for creating our QA pairs, the architecture of our CA and the details and results for the evaluation are then presented in Chapter 5. Chapter 6 discusses the previously presented results as well as any limitations of our agent. Chapter 7 finally provides a conclusion of this thesis and suggests directions for future work.

2. Foundations and Theoretical Background

This chapter lays the foundation to understand the subsequent chapters. First, it clarifies different terms related to conversational agents, discusses popular development platforms for CAs and addresses the challenges of creating an agent for consumers in the health context. Secondly, we present in-depth background information on two question-answering approaches that are used by the final agent.

2.1. Conversational Agents

2.1.1. Fundamentals and Definitions

Before diving into the workings of a CA, we will introduce some important definitions to avoid any ambiguity. We are following the definitions provided by Allouch et al. [4], who begin with an introduction into the most general concept of a *dialogue system*. It describes a system which uses natural language to communicate between human and machine. A *conversational agent* is a dialogue system which can generate and understand natural language responses using any medium, e.g. text or voice input as well as output. For instance, a telephone hotline, where you are instructed to say "one" if you want to talk to an English-speaking assistant or "two" if you want to talk to a German-speaking assistant, represents a basic dialogue system. This cannot be considered a CA because the user is not responding in natural language. CAs are further categorized into text-based and voice-based agents. The former is interacting with the user through a chat while the latter uses voice for in- and output. Some agents may use a combination of both, e.g. voice-based input but output through text, although Allouch et al. do not specify this as a subclass of dialogue systems. However, they also define *embodied agents* as another category of CAs which use gestures in addition to text- or voice-based communication. As this is not of relevance to this thesis, we do not go into detail here. All definitions are summarized in Figure 2.1, allowing us to classify the agent developed in the scope of this thesis as a voice-based conversational agent.

After clarifying the most important terms, we can now outline the general architecture of a CA and the additional components needed to make a CA become a voice-based agent. At the core all CAs work in the same ways: They consist of a natural language understanding (NLU) component, a dialogue manager (DM) and a natural language generator (NLG) component [4]. The NLU component interprets the user input and converts it into a semantic

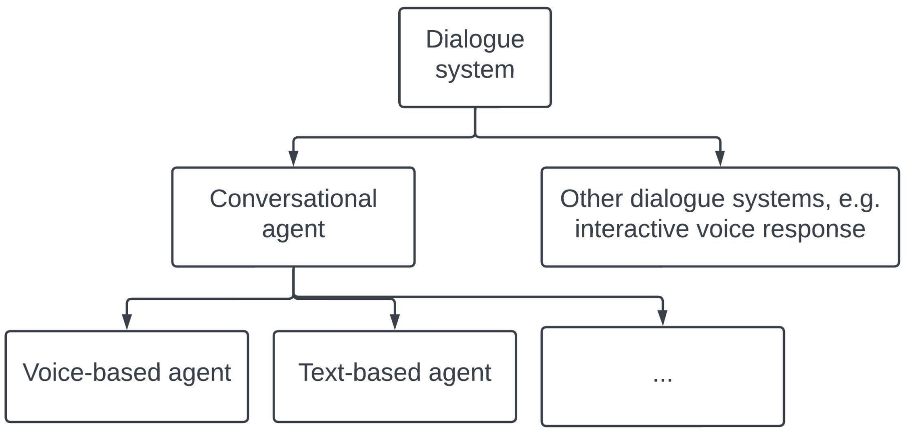


Figure 2.1.: Hierarchy of different definitions related to conversational agents, based on [4]

representation which can be further processed by the DM [5]. The DM decides how to respond to the processed user input by identifying the user's motivation. A widespread concept for this process is the use of so-called *intents*. One agent usually has multiple intents which help the agent to map different scenarios to specific actions. This is achieved through identification of specific keywords previously extracted by the NLU component. The DM recognizes those keywords and matches them with one of its intents [6].

For instance, consider a conversational agent designed to perform the two actions of informing the user about either the weather forecast in Munich or the current time. This agent contains two intents, mapping different inputs to one of the two actions. The first intent might trigger for keywords such as "weather" or "rain" in order to match questions like "How is the weather?" and "Will it rain today?" to the action of providing weather information. Vice versa, the DM interprets keywords like "time" or "late" as indicators to execute the action of informing about the current time.

While this example is kept basic to help understand the fundamental mechanism behind an intent, it should be noted that agents nowadays often employ machine learning techniques to correctly assign inputs to the agent's intents, making the process more complex.

Additionally, the DM preserves a chat history to help with creating a suitable response. Finally, the DM generates a semantic response which will then be transformed into natural language by the NLG component.

The key difference between voice- and text-based agents lies in two additional components required by the voice-based agent: A speech-to-text (STT) component and a text-to-speech (TTS) component. The first component is used for translating the user input to text to be then further processed by the NLU component. The TTS component receives the response from the NLG component and then translates it into a voice-based output. The resulting architecture of a voice-based conversational agent can be seen in Figure 2.2.

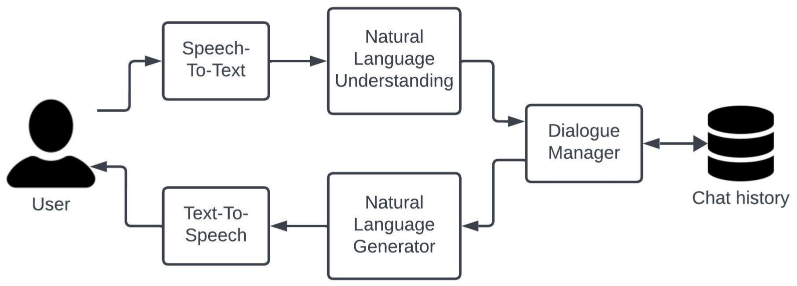


Figure 2.2.: General design of a voice-based conversational agent, based on [4]

2.1.2. Development of a Conversational Agent

Several companies offer platforms and frameworks to support developers in creating their own CAs, such as Amazon Lex, Microsoft Bot Framework, Watson Assistant by IBM, Dialogflow by Google, and Rasa. In this section we compare advantages and disadvantages of Dialogflow and Rasa. Dialogflow as the selected platform for developing the agent of this thesis after evaluating different options and Rasa as a commonly used open-source alternative. For a more detailed comparison of all the platforms mentioned above refer to Dagkoulis and Moussiades [7], where Dialogflow and Microsoft receive the best ratings. Nevertheless, the ratings should be interpreted carefully since the best suiting platform highly depends on the specific use-case.

Dialogflow, a Google-owned closed-source platform, offers two versions: Dialogflow ES and Dialogflow CX. The former is a lighter and more cost-effective option, while the latter is designed for more complex and larger agents, providing additional features to manage your agents. At its core, Dialogflow extracts the entities in user requests and then matches the request with one of the agent's intents, like what we have introduced in the context of conversational agents and their general architecture. Furthermore, Dialogflow uses contexts to track the conversation history and stores data which might be needed in future steps of the conversation. One example where such a context is essential is a flight booking chatbot: At the beginning of the conversation the user is asked for the departure and destination airports. In further steps, the chatbot requests personal details and payment information but must remember the flight's departure and destination. This challenge could be solved by setting a context that stores this information while continuing with the conversation. This context can additionally help with determining the intent, as the chatbot knows that if the context with the flight's departure and destination airport is given, it does not need to match to an intent which asks for this data again. Instead, the chatbot can e.g., match to an intent that asks for the personal details.

One of Dialogflow's main advantages is the ease of use. Dialogflow offers an intuitive graphifcal user interface (GUI) as well as prebuilt agents for common use-cases. It supports up to 123 languages and includes speech-to-text as well as text-to-speech components. Additionally, a comprehensive online documentation and wide range of online community resources are available. However, there are also disadvantages to using Dialogflow, including that it cannot be run on-premises, because it is a Google product and operates exclusively on the Google Cloud. Another drawback is the high price of Dialogflow, even for the ES edition, especially when comparing it to an open-source platform.

This leads to the discussion of the second platform: Rasa. Rasa not only offers an open-source edition but also an enterprise version. The enterprise version focuses on supporting customers with the deployment of their own CAs, for example, by providing a managed service for running the customer's Rasa application. Additionally, it extends the open-source version by various AI features including an LLM-native approach. As with Dialogflow, Rasa is extracting the entities from an incoming prompt and matches it to an intent using Rasa's NLU and dialogue management component. Being open-source is a key benefit of Rasa which leads to more advantages. For instance, Rasa can be deployed on-premises and is free of charge as long as you do not use the enterprise version. Rasa is highly customizable which is advantageous for experienced developers who are looking for fine-grained control over their CAs. Nevertheless, for developers new to this field, this can be a disadvantage, as it can become very time-consuming and challenging to create a conversational agent.

In conclusion, Dialogflow is a more suitable choice for developing our agent as our agent's architecture is simple and therefore does not require the customizability offered by Rasa. Neither do we require to deploy our agent on-premises. Therefore, we can benefit from the ease-of-use and many features that Dialogflow provides. Additionally, the ALPHA-KI project uses Dialogflow as their platform which makes it easy to integrate the agent presented in this thesis into their existing systems.

2.1.3. Conversational Agents in Healthcare

CAs find application in the medical domain for various different use-cases. Therefore, Montenegro et al. [8] have created a taxonomy to give a better overview of the different fields of application. This taxonomy mentions many key areas where these agents are making an impact, including the training of medical students, aiding in suicide prevention, or supporting physicians in diagnosing diseases based on symptoms presented by patients. These examples represent only a few of the numerous fields where agents are being used, showcasing the diverse potential of conversational agents in healthcare.

A particularly unique field is answering questions of consumers who are unfamiliar with medical terminology. As explained by Zweigenbaum [2], this area of QA can be even more challenging than open-domain QA, i.e. responding to questions about a wide range of general topics. A significant challenge when it comes to CHQs is the gap between medical jargon and consumer language.

For instance, let us consider an agent that searches through a collection of documents on different health-related topics in order to respond to incoming questions based on one of the documents. Let us assume, in the collection exists a document that discusses concussions but refers to them as "mild traumatic brain injury (mTBI)", a term more commonly used by physicians. If a user asks, "What can trigger a concussion?", our system must map this question to mentioned document which might never use the term *concussion* but always writes about mTBI. Although the answer might be available in this document, the QA system may struggle to find it. If the system does determine the correct document, it still must return a response which can be understood by the user. This possibly involves an explanation that concussion and mTBI are the same thing or maybe replacing the term *mTBI* with *concussion*.

Furthermore, the agent will be deployed onto the watches from the ALPHA-KI project, which are used in healthcare, primarily by elderly people. They are not only likely to be unfamiliar with medical terms but also they do not have the same access to online resources, e.g. Wikipedia, as the younger generation. Thus, it is crucial to avoid any medical terminology, keeping in mind that the users are unlikely to look up unfamiliar terms on the internet.

2.2. Pre-Trained Language Models

Pre-trained language models represent a specific class of machine learning models which were trained in advance and are ready for immediate use. HuggingFace[1] is a platform that provides many open-source models including pre-trained language models. They have been trained on extensive amount of data, making them robust and suitable for many different applications.

Transformers are a specific type of architecture that builds the foundation of most modern pre-trained language models, first introduced in the paper "Attention is All you Need" by Vaswani et al. [9]. In order to train a model on such a huge amount of data efficiently, self-supervised learning strategies are applied, enabling them to learn from the data without human intervention. Not depending on a human is essential, as it is not feasible with the amount of data that is required to train a transformer. In the following we explain two different types of transformer models, that are used in this thesis and explain how they enable self-supervised learning strategies. [10]

[1] https://huggingface.co/

2.2.1. Auto-Encoding Transformers

First, we discuss *auto-encoding transformers* which are highly important to our agent, as we use such a model to match an incoming question to one of the questions stored in our database. These transformer models are commonly used for question-answering applications but also for sentence classification or named entity recognition.

The main component of an auto-encoding transformer is the encoder part. It is responsible for translating input text into a contextualized representation, i.e. usually a vector representing the meaning of the input text which takes the context into account [11]. The self-supervised learning strategy mainly applied to train auto-encoding models uses so-called *masked language modeling*: With every training step the model is presented with sentences in natural language where a random word is masked, and the model must predict the masked word. This technique is considered self-supervised as the prediction of the model can be directly compared to the word that has been masked in order to check if the prediction was correct. A key aspect of such an encoder is its ability to take words preceding and following the masked word into account when predicting the masked word. Through training of an auto-encoding model on a large corpus of data, it learns grammar and sentence structures only by predicting masked words. [10]

Efficiently Learning an Encoder that Classifies Token Replacements Accurately

One well-known example of an auto-encoding transformer is BERT which we use when performing the similarity search. Moreover, we use another auto-regressive model, called "Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA)", whose training method slightly differs from the masked language modeling approach as it trains two separate models: a generator and a discriminator. The generator is trained to replace tokens in a given text with plausible alternatives. The main model, which is the discriminator, is then trained to identify the "fake" tokens created by the generator. This method is more efficient, especially when computational resources are limited [12]. ELECTRA models are commonly used for extractive question-answering tasks as the model that we decided to use in this thesis. It was further fine-tuned on a question-answering dataset, which leads to further performance increase in extracting the correct answer.

Both models that we use are published by *Deutsche Telekom AG*[2] on HuggingFace and fine-tuned with German data.

2.2.2. Auto-Regressive Transformers

The second category of transformers is utilized for the construction of our dataset of QA pairs, namely *auto-regressive transformers*. They are commonly used for text generation tasks such as creating QA pairs. Unlike auto-encoding transformers, auto-regressive models only use preceding words for generating predictions. The self-supervised learning strategy employed by this model family is the following: Initially, the model receives the start of a piece of text, and its task is to predict the subsequent word. The model's

[2]Deutsche Telekom AG on HuggingFace: `https://huggingface.co/deutsche-telekom`

prediction is then compared with the actual next word from the text, providing immediate feedback on the accuracy of its guess. This process is repeated word by word for a vast number of texts until the model learns to generate grammatically correct sentences. This method allows the model to generate text by considering each preceding word including the words already generated. ChatGPT is a well-known example of such an auto-regressive transformer that we also use for generating our QA pairs. [10]

2.3. Question-Answering

In this section we present two different approaches on QA which are commonly used by conversational agents. The first method is a similarity-based approach as already introduced in the previous chapter. The second strategy is extracting the response from a collection of documents, a process known as information retrieval (IR) [1]. After presenting technical insights to both approaches, we highlight the differences between them to avoid any ambiguities in the coming chapters.

2.3.1. Similarity Search

Similarity search describes the process of finding the most similar item for a given query item. In the context of this thesis this means having an incoming question from a user and retrieving the most similar question in a large database of questions. This large database contains questions as well as their corresponding answers. Thus, if we find a highly similar question in this database, we can answer the user's question simply by responding with the corresponding answer to the matched question. The main challenge remains in finding a suitable measure for comparing two questions, or more general two sentences. Additionally, we aim to find that match in a feasible time which also becomes challenging regarding the fact that we might have a large corpus of QA pairs in our database. Therefore, in this section we will introduce the concept of embeddings as well as present three different similarity metrics.

In order to be able to compare two sentences and measure their similarity it makes sense to map those onto a vector space. Numerical vectors have the convenient property of being easily comparable with high efficiency: Vectors which are close to each other are similar and vice versa. Thus, if we map our questions to a vector in the same vector space, we can apply different similarity metrics. That is, what *embeddings* are used for.

Word embeddings in the way we use it in this thesis were first introduced approximately 20 years ago by Bengio et al. [13] as a solution for the curse of dimensionality. This was due to former approaches of embedding text as vectors had a high correlation between the number of dimensions and the number of words in the existing vocabulary. Vocabulary here describes the set of unique words or terms present throughout all texts used for the similarity search. In our case that would be the unique words contained in all questions stored in our database.

A key advantage of embeddings is their ability to take semantic meaning into account. This is important because if we would compare two words solely based on characters this can lead to misleading results. For instance, if we consider the two words "cat" and "kitty" which describe the same thing, it becomes difficult for character-based approaches to identify their similarity.

To achieve a mapping of text to vectors which maintains the semantic meaning of the text, we use a BERT-based transformer published on HuggingFace by Deutsche Telekom AG[3]. It takes any text-string as input and returns a 1,024-dimensional vector.

Once we have computed the embeddings for two questions, we still need a suitable metric for identifying similar vectors. As mentioned before, semantically similar text will be represented by vectors which are close to each other in the resulting vector space. Therefore, we aim to measure the distance between two vectors in order to compute the similarity. Popular choices for computing the distance between two embeddings are cosine similarity as well as Euclidean or Manhattan distance.

Cosine similarity measures the cosine of the angle between two vectors in order to compute their similarity. It is defined by

$$S_C(u,v) = \frac{u \cdot v}{\|u\|\|v\|} \tag{2.1}$$

where u and v are two vectors of the same length and the dot product of a vector x with itself is defined as $\|x\|$ [14]. The cosine similarity only takes on values in the range of $[-1, 1]$ with $S_C = 1$ meaning the two compared vectors are equal and $S_C = 0$ indicates two vectors are orthogonal and therefore the corresponding sentences very different. Computing the cosine similarity is equal to computing the dot product of two vectors with unit length [15]. Let us assume vectors u and v are of length 1, which means $\|u\| = \|v\| = 1$, we can then simplify (2.1) as follows

$$S_C(u,v) = \frac{u \cdot v}{\|u\|\|v\|} = u \cdot v \tag{2.2}$$

Therefore, cosine similarity is very fast at comparing large amounts of vectors: It is a single operation which can be perfectly parallelized.

Another popular metric for distance measuring of two vectors in the context of word embeddings are Euclidean and Manhattan distance. Both consider a vector's magnitude in contrast to cosine similarity which only measures the relative position of both input vectors. Thus, Euclidean and Manhattan distance return an absolute value instead of a relative value as with cosine similarity. Euclidean distance can be defined as

$$D_E(u,v) = \sqrt{\sum_{i=1}^{n} |u_i - v_i|^2} \tag{2.3}$$

[3]Transformer for embeddings: `https://huggingface.co/deutsche-telekom/gbert-large-paraphrase-cosine`

with vectors u and v being of length n. Manhattan distance can be written as

$$D_{Man}(u,v) = \sum_{i=1}^{n} |u_i - v_i| \tag{2.4}$$

One might have noticed the similarites both metrics share: They are both variants of the Minkowski distance.

$$D_{Min}(u,v) = (\sum_{i=1}^{n} |u_i - v_i|^p)^{\frac{1}{p}}, p \in \mathbb{N} \tag{2.5}$$

If we set $p = 1$ we retrieve D_{Man}, whereas for $p = 2$ the Minkowski distance equates to our definition of the Euclidean distance.

The decision which metric to use for comparing word embeddings, highly depends on the data, the use-case and the model used to compute the embeddings. After comparing the performance of each metric, we retrieved satisfying results for all of them. Based on these findings and the requirement of our final system to be highly efficient in comparing a large number of vectors, cosine similarity seemed an appropriate choice.

Furthermore, we found comparative analysis which supported our decision, as cosine similarity was the best performing metric [16, 17]. Although the setting and use-case of both studies do not perfectly reflect our application, they still indicate that cosine similarity is a valid choice. In the end, our decision mainly based on the provided efficiency.

2.3.2. Information Retrieval

Search engines are most likely the primary field of application for IR. IR can be defined as finding the best fitting piece of information for a given query in a large corpus of documents. In the case of search engines, the large corpus of documents are all publicly available web pages, and the query is given by the user. In our case, IR is more relevant in the context of answering a user's question to a health-related topic. We need to find a document which best answers the user's question from a collection of documents on different topics. Then we extract a passage from the retrieved document that answers the question.

This approach is employed by our agent's QA pipeline. The idea is to use this as a fallback method if the similarity-based approach failed to answer the user's question. Therefore, the thesis focuses on the similarity-based approach. Nevertheless, in the following sections we explain the fundamentals about traditional IR which are the basics to understand the full pipeline presented in subsequent chapters.

One key algorithm when it comes to information retrieval is BM25. It is used by search engines for retrieving a ranked list of results for a given query and plays a big role in QA. Robertson and Walker [18] were the first to introduce BM25 in 1994. The algorithm tries to find a document from a collection of documents which matches best to a given query. Although this ranking function is used by QA systems, the original problem statement is not related to question-answering, since the query does not necessarily need to be a natural language question or statement. During the past 30 years many variants have been developed, but the main concepts remain the same: term frequency, term saturation, inverse document frequency (IDF) and normalization.

Term frequency is the first concept and simply counts how often a term is repeated in a document. Documents containing the terms used in the query more frequently are considered a better match.

However, the more times a query term appears in a document the less impact each additional occurrence has on the overall score, what is called *term saturation*. There are a few reasons why this is essential in order to retrieve the best matching document. To illustrate, let us assume we have a document A which contains one word from our query 50 times and another document B which contains the same word 100 times. We would agree that B is more relevant to the query, but probably not twice as relevant as A [19].

The next concept, IDF, helps with weighting terms as less important which frequently occur in any document. It does that by counting the number of times a word is shown throughout all documents which we have stored. Therefore, common words like "and", "or", "if", etc. receive lower importance since they are usually frequently used in all of the documents.

Lastly, *normalization* describes taking the length of each document into account in addition to term frequency. This particularly makes sense if we consider the following example. In our database are two documents: document A which contains 500 words in total including 100 times the word "pink", document B which contains 100 words mentioning "pink" 90 times. If the input query is "pink", document B should be considered more relevant due to the higher frequency in relation to the length of the document. Therefore, BM25 puts occurrences of query terms in relation to the total length of each document. For further details, especially on mathematical background of BM25, refer to Amati [20].

Since we are in the context of QA, after retrieving one or multiple candidate documents, we still need to identify a passage from our documents which we use to respond to the question. A common approach is to use neural networks which take a question and a document as an input and return a few words up to a few sentences from that document to answer the given question. One can either decide to train their own model or use a publicly available pre-trained model to accomplish this task. Training your own model has the advantage of being able to fine-tune it to your specific needs or maybe a certain vocabulary used in the given context. However, the main drawback of training your own model are the resources it requires, not only time but also financial and computing resources.

We decided to use a pre-trained ELECTRA from HuggingFace[4] which takes a question and a text as an input and returns a passage which best answers the question. In addition to the answer, it provides a confidence score, indicating how likely the response answers the question. As the model achieved satisfying results when experimenting with a few

[4]ELECTRA model for QA: `https://huggingface.co/deutsche-telekom/electra-base-de-squad2`

sample queries, we decided to integrate it into our traditional IR approach. Otherwise, one can consider training their own neural network or employ a different approach.

2.3.3. Similarity Search vs Information Retrieval

Throughout this chapter we can notice that similarity search and information retrieval share similar goals and concepts. BM25, a method introduced in the context of IR, can also be used to search for similar items as in similarity search. Accordingly, embeddings and cosine similarity, typically associated with similarity-search, can be used in IR to retrieve matching documents. In Chapter 5, the presenting of our results, we will see that embeddings in the context of IR are used as well.

Following the definition of Welivita and Pu [1], the main difference between those two approaches on QA lies in the idea of how to store the data which will be queried. For similarity-search, a large number of QA pairs is stored in a database to then perform different methods to identify the best matching question. In contrast, for IR a database is populated with raw documents. The database is then queried to return one or multiple candidate documents from which a passage is extracted that answers the incoming question.

Both approaches offer flexibility in employing many more techniques besides embeddings and BM25 to answer a question. As previously mentioned, there are no restrictions or rules which similarity metrics or algorithms to use for each approach. We decided to explain each method in the context of the approach where it is typically used. Furthermore, the techniques introduced in this section were selected because they are widely applied, and they are employed by our agent.

3. Related Work

In this chapter we provide an in-depth review of work related to this thesis. According to our research questions, this chapter covers three topics. The first part of this chapter concentrates on the development and evaluation of a CA and outlines different approaches for consumer health QA. In the following part, we present different methods used in literature to evaluate QA agents. Finally, the last part outlines different existing approaches on creating a dataset of QA pairs which can be used by a similarity-based approach. Although the three correlate with RQ2, RQ3 and RQ4, their order is slightly changed, according to the decision process: First, a suitable approach for QA and for the agent was chosen which was the similarity-based approach and afterwards we decided how to evaluate the agent and how to construct a suitable dataset for the CA.

3.1. Conversational Agents in Consumer Health Domain

3.1.1. Existing Question-Answering Approaches

In literature we can find several approaches on classifying existing QA systems, for example Soares and Parreiras [21] came up with four different categories. While they took QA systems for any domain into account, Welivita and Pu [1] focused on existing QA systems in the consumer health domain. They introduced three main categories: Traditional IR-based, knowledge graph-based and entailment-/similarity-based approaches. These categories are similar to the ones presented by Soares and Parreiras. Additionally, both literature reviews introduce hybrid approaches which have integrated at least two different methods in their system. In the following we will show three CAs each representing a different category.

Wang and Nyberg [22] presented an architecture which solely works with traditional IR. The system is divided in three steps: Clue retrieval, answer ranking and answer passage tiling. The first step queries a collection of documents simply by using search engines, such as Bing Web Search and Yahoo! Answers and retrieves web pages related to the question. The next step then ranks those candidate answers from the first step using a weighted combination of optimized BM25 similarity scoring and two different neural networks for estimating the relevance of each document. The first neural network uses the document's title and the second one the document's text to predict the relevance. The final step then concatenates the highest ranked answers which results in the system's final answer.

Even though Yang et al. [23] implemented a hybrid approach which uses a combination of a knowledge graph and traditional IR, it leads to a good understanding on how knowledge graphs are employed for QA. The system organizes knowledge about each medical entity as a tree with the entity being the root element and its attributes the leaves. The entities are stored in an array and based on the question's focus and type the corresponding entity is identified. The focus describes the extracted entity, e.g. hypertension. Additionally, Yang et al. defined 23 question types, such as cause or treatment. Based on the question type the correct leaf node of an entity can be retrieved.

The recognizing question entailment (RQE)-approach used by Abacha and Demner-Fushman [24] relies on the concept of entailment: If a user asks a question A, they try to find a question B which is entailed by question A, i.e. a question whose corresponding response is also a response to question A. Assuming one finds such a question B, the system can respond with the corresponding response to B. This approach therefore stores a large amount of QA pairs and tries to find such a question which is entailed by the input question. There are two main challenges with this approach: identifying an entailed question and retrieving a set of similar questions.

Additionally, Demner-Fushman et al. [25] published the Consumer Health Information and Question Answering (CHiQA) system which employs a hybrid approach: a component using traditional IR and a component reusing the RQE method. Both components generate candidate answers from which five answers are chosen to show to the user.

Welivita and Pu [1] differentiate between *single-turn* and *multi-turn* QA: The former answers only single questions while the latter answers a question through a conversation where the user might ask follow-up questions, coreferencing information from previous messages.

Each of the presented approaches implements a single-turn agent that only handles a single question at a time. A multi-turn agent, called *enquireMe*, was published by Wong et al. [26]. At its core it employs a similarity-based approach. They used QA pairs extracted from question-and-answer websites as *Yahoo! Answers* and *Answers.com* as their source of information. When a user inputs any text, the system extracts key phrases and weights them according to importance for extracting the meaning of the input. Those weighted key phrases are then used to query the QA pairs for similar questions. The retrieved questions will then be ranked based on different metrics and the answer corresponding to the highest ranked question will be returned to the user. The conversational nature of this system is implemented by the system remembering previous key phrases and adjusting weights if there are key phrases which appear in the most recent input as well as in previous inputs. Thus, the system adjusts the weights in favor of those key phrases.

3.1.2. Evaluation

The Text REtrieval Conference (TREC) is an annual event and series of evaluation workshops designed to compare the latest results in research in the field of IR and QA. In 2017

TREC organized a medical subtask to evaluate QA systems which specialized on the field of consumer health questions [27]. The questions presented here are publicly available[1] and used by many studies to evaluate their resulting systems [22, 23, 24, 28]. At the conference the responses of the evaluated CA were assessed on a Likert scale with the four levels correct and complete (4), correct but incomplete (3), incorrect but related (2) and incorrect (1) [1].

Unfortunately, as with all publicly available datasets containing CHQs which we found when researching, the questions from TREC 2017 are in English. Thus, they cannot be used for evaluating our agent and we therefore came up with our own evaluation methods as described in Chapter 5.

Furthermore, since there are no public datasets on evaluating conversational QA systems, Wong et al. [26] obtained questions from the website WebMD [29] in the form of "What is lung cancer?" and extended them by adding follow-up questions as e.g. "What causes it?" and "What are its treatments?". This made it possible to evaluate the system's ability of resolving coreferences.

In contrast, our agent is more conversational in the way of asking follow-up questions itself. It does so either to overcome ambiguity in the user's question or to recommend new questions, engaging the user in continuing the conversation. Therefore, this evaluation method is not applicable to our agent.

3.2. Generation of Question-Answering Pairs

As we have decided to employ a similarity-based approach, we require a dataset of QA pairs which builds the foundation of the resulting CA. Therefore, the overall accuracy in answering a question highly depends on the quality of the dataset.

A simple approach to obtain a dataset is to reuse an existing, publicly available set of QA pairs, for example the MedQuAD which contains 47,457 medical QA pairs [24] which is referenced by a few studies [24, 30, 25]. However, this dataset is an exception as generally there is a lack of high quality medical QA datasets [3]. Furthermore, in this thesis we require a dataset of German QA pairs which additionally needs to be designed for voice-based output, i.e. not allowing answers longer than two sentences.

Since, to the best of our knowledge, there do not exist any datasets fulfilling those requirements, we decided to generate our own QA pairs. Some papers scrape different QA forums as Yahoo! Answers for creating such a database [26, 31]. Similarly, the previously mentioned MedQuAD dataset was also automatically generated by crawling various trusted

[1] `https://github.com/abachaa/LiveQA_MedicalTask_TREC2017`

medical websites, e.g. National Cancer Institute[2] or Genetic and Rare Diseases Information Center[3] and parsing them into handcrafted patterns [24].

In other domains than the health domain one can find approaches on automatically generating QA pairs using an LLM. Kalpakchi and Boye [32] automatically generated a dataset of open-domain multiple-choice questions using GPT-3. Samuel et al. [33] presented a system which first synthesizes a new context and then uses the synthetic context to automatically generate new questions. This approach was designed for areas with sparse online resources available, including covid as a health-related topic.

3.3. Novelty of Our Approach

When reviewing literature on the creation of a QA system for CHQs, one can find a few different approaches especially motivated through the medical subtask at the TREC 2017. If one limits the search further to a similarity-based approach, only two or three conversational agents can be found. To the best of our knowledge, setting up a pipeline for QA which mainly bases on a similarity-based approach that uses a BERT model to compute embeddings, was not developed by any previous research. Furthermore, although our agent might not handle coreferences as does enquireMe [26], the agent's QA pipeline poses follow-up questions for disambiguation, employs traditional IR as a fallback strategy and recommends new questions to keep the user engaged. These features, which contribute to a more conversational system, have not been integrated into any existing QA systems in consumer health domain, as they solely focus on answering a user's question.

Additionally, within our domain, our research did not find any existing literature on creating a dataset of QA pairs based on data crawled from online resources using a LLM, as of November 2023. Therefore, regarding the implementation of a CA using a similarity-based approach with an automatically generated dataset of QA pairs, it seems we are the first to develop such a system. With this contribution we aim to initiate more in-depth research in this direction.

[2]National Cancer Institute: `https://www.cancer.gov/types`
[3]Genetic and Rare Diseases Information Center: `https://rarediseases.info.nih.gov/`

4. Methods

This chapter briefly introduces the methods applied for answering each research question. According to the research questions we have outlined four major steps, as illustrated in Figure 4.1.

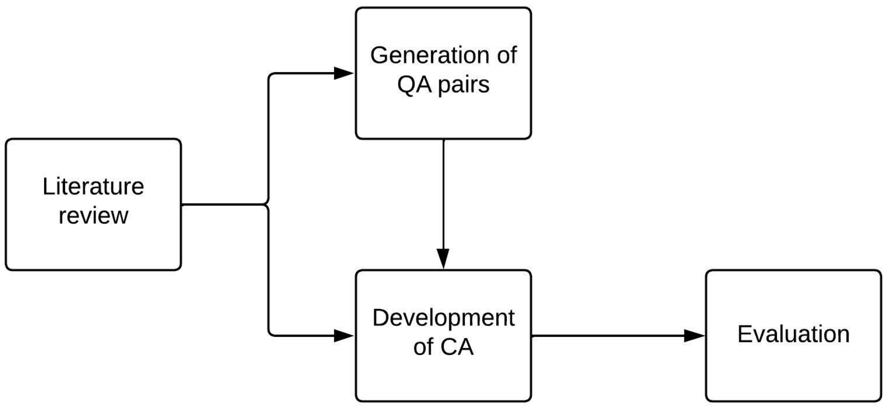


Figure 4.1.: Overview of different steps performed to answer each research question

4.1. Literature Review

In this section we explain our methods for the first step of this thesis which focused on exploring existing approaches in literature. Therefore, we systematically searched for any scientific resources related to our topics as this step set the foundation for answering the other research questions.

We divided our research into two separate steps: First finding existing approaches on automatically generating QA pairs using an LLM and second building a CA in the consumer health context. For both steps, our primary research tool was Google Scholar[1]. Additionally, we used backward search, i.e. reviewing references cited in each paper found.

For the first part, we performed the following queries, annotated with the number of results:

- "question answer generation" "health" "llm" (9 results)

[1]Google Scholar: `https://scholar.google.com/`

- "question answer generation" "llm" (51 results)
- "qa pair generation" "llm" (10 results)
- "generating qa pairs" "llm" (14 results)
- "creating qa pairs" "llm" (2 results)

The query always contained information about the desired action, i.e. generating question answer pairs, as well as information about the technology to use which in our case is an LLM. In the beginning of our research, we have limited our research to the health domain, which led to only nine results with one paper being relevant. Because of the lack of literature, we broadened our search to include QA dataset generation across all domains. The resulting four queries returned on average 19,25 results where we selected nine based on title and abstract to be further investigated.

We also explored alternative approaches than using an LLM to generate QA pairs. We applied the two search strings *"question answering" "health"* and *"consumer question answer" "health"* but found no useful studies. However, in the second part of our literature review, when analyzing existing solutions on QA in the consumer health domain, one relevant paper introducing an alternative approach was retrieved.

Overall, we have identified four helpful papers on generating QA pairs helping us to determine an effective prompt design and understanding the impact of different hyperparameters when using GPT-3.5-Turbo for text generation. Unfortunately, one of the papers is not accessible online which reduces the number of relevant results to three.

For the second part of this literature review, the following were our performed queries with the corresponding number of results:

- "conversational question answering" "health" (457 results)
- "conversational agent" "health" (12,000 results)
- "question answer" "health" (48,300 results)
- "consumer question answering" "health" (43 results)

We structured the query as a combination of the type of system we want to build, e.g. "question answering", and the domain we are interested in, namely the health domain. As the number of results was too large to read through all of them, we decided to only take the top 20 results per query into account.

Based on our pre-selection, we could identify four highly relevant surveys presenting various approaches on conversational agents in the health domain. From here we used backwards search to read through all the presented CAs that covered the consumer health domain. In total those were 19 candidate agents that we investigated in detail and a number of seven that were relevant to this thesis. Those agents helped us especially with identifying the challenges and advantages of different solutions including the similarity-based approach that our agent mainly relies on.

4.2. Generation of Question-Answering Pairs

After our in-depth review of existing solutions, we started with generating the dataset of QA pairs. The pipeline used for automating this process is shown in Figure 4.2.

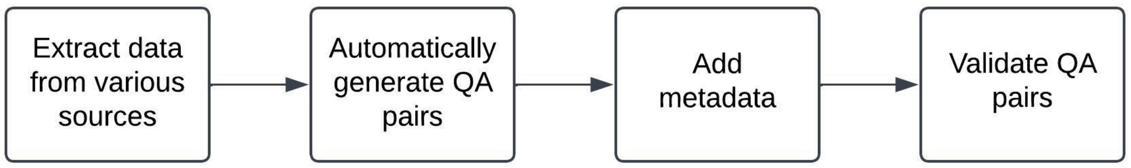

Extract data from various sources
Automatically generate QA pairs
Add metadata
Validate QA pairs

Figure 4.2.: Pipeline for automatic generation of QA pairs

The first step of the pipeline was to collect data from various sources. The first source is the German health magazine *Apothekenumschau*[2] which publishes articles about a diverse range of health topics, usually as a printed magazine. However, they also publish articles online on their website which they ensure to validate by health experts.

Another source used for constructing our dataset were PDF documents, published by three different German health organizations. The first, *Deutsche Hochdruckliga e.V. - Deutsche Gesellschaft für Hypertonie und Prävention*[3], aims to support research and prevention in the field of high blood pressure. The second organization we conducted PDF files from, *Nationale VersorgungsLeitlinie*, is an initiative formed by various professional societies, healthcare organizations and the German government providing information for patients and the public. Lastly, we used data from *ANS-Ambulanz der Uniklinik RWTH Aachen*[4], a clinic associated with *RWTH Aachen University*.

After collecting the data, we processed it through an LLM in the second step of our pipeline. We decided to use GPT-3.5-Turbo by *OpenAI*[5] because it is one of the leading LLMs and provides an application programming interface (API) to send your requests to. Therefore, with this choice, it is not required to deploy our own model, saving both resources and time.

[2]Apothekenumschau: `https://www.apotheken-umschau.de/`
[3]Deutsche Hochdruckliga e.V. - Deutsche Gesellschaft für Hypertonie und Prävention: https://www.hochdruckliga.de/
[4]ANS-Ambulanz: `https://www.ukaachen.de/kliniken-institute/ans-ambulanz/die-ans-ambulanz/`
[5]OpenAI: `https://openai.com/`

Next, we add some metadata to our QA pairs and store them in a database. In the last step of our pipeline, we validated our QA pairs by sending them to several health experts which review and annotate all the QA pairs. This feedback will also be presented in the following chapter as it provides valuable insights on the quality of our resulting dataset.

4.3. Development of a Conversational Agent

In parallel with the construction of the dataset of QA pairs, we started to develop our agent and integrated the dataset as its knowledge base once it was finished. As evaluated in Chapter 2, Dialogflow was our choice for developing the agent. Additionally, we set up a Python Flask app to serve as a webhook service. This service, to which Dialogflow redirects all incoming requests, allows to implement more complicated logic. In this Flask application, we integrated a whole pipeline of steps to process, analyze and answer incoming questions. At its core, the process used the similarity-based approach with our constructed dataset but additionally, a traditional IR approach was implemented as a fallback method.

4.4. Evaluation

The last step was the evaluation of the resulting system. We split the evaluation into two parts: an automatic evaluation and conducting qualitative user feedback. For the automatic evaluation, we selected a subset of our QA pairs, slightly changed the wording using an LLM and measured how often our agent could correctly match the changed questions to the original ones. This approach aims to evaluate the matching performance of our agent.

Furthermore, we asked a group of test users to interact with our final agent. Afterwards, they filled out a form which we then analyzed. This second evaluation strategy aims at assessing conversational features, such as the ability to handle ambiguous questions and to engage the user in further interactions after the initial question was answered.

5. Results

5.1. RQ1: Literature Review

This section answers the first research question *"What existing approaches are there in conversational question answering in context of health-related topics?* by addressing the papers selected from our literature research in more detail. As explained in Chapter 4, we divided our research into two parts. We start with presenting methods on using LLMs to automatically generate QA pairs, before moving on to approaches for conversational agents in the health domain.

5.1.1. Generation of Question-Answering Pairs

As already mentioned in Chapter 3, it was difficult to find any studies that focus on automatically constructing a dataset of QA pairs using LLMs. As we aim to learn from the challenges and decisions of others, we decided not to restrict ourselves to only health-related approaches for this section.

Samuel et al. [33] generated questions across three different domains: Healthcare, public policy, and technology. Their main idea was augmenting data in low resource fields to then generate QA pairs which can be used to improve the performance of QA systems. As we are not looking to augment contexts, the study is not exactly reflecting our use-case. However, the approach gives valuable insights on how to enable few-shot learning for generating QA pairs, i.e. a strategy for improving the results of an LLM by providing example outputs.

The second interesting finding was published by Kalpakchi and Boye [32] who generated Swedish multiple-choice questions for open-domain topics. They sent texts from the national tests of Swedish for Immigrants courses to GPT-3 to generate multiple-choice questions. For each question, GPT-3 was instructed to create four different answers where the first answer should always be the only correct option.

Although they decided to use zero-shot learning, i.e. not providing any examples to the LLM on how to respond, a key takeaway from this paper was the prompt design. They instructed GPT-3 exactly on how many questions to generate based on the length of a text: the longer the text the more questions were asked for in the prompt. The model produced for 89.6% of the prompts the desired number of questions. Mainly for long texts it produced less questions than asked for. This underpins our decision to send single paragraphs of text to the model instead of the whole article.

Another interesting insight is on ChatGPT’s ability to generate questions in another language than English, which in this case was Swedish. GPT-3, the model used in this study, was trained on 92.6% English data and only 0.11% Swedish data and still, the results from this paper were satisfying. In this thesis, we use the GPT-3.5-Turbo model, and although there is no detailed information on the model’s language training distribution, it most likely was not trained on much German data. Thus, the results from Kalpakchi and Boye indicate that using GPT-3.5-Turbo in another language than English can lead to satisfying results.

Furthermore, for generating the questions, the hyperparameters of GPT-3 were set to default, a setting we also applied when generating our QA pairs. Hyperparameters are configurations available when interacting with GPT through the API that influence GPT’s behavior. Kalpakchi and Boye explained that the choice of hyperparameters is a trade-off between correctness and creativity: If the values for different hyperparameters get maximized, they lead to more creative answers but at the same time more creativity can result in the model "drifting away" from the original text. According to Kalpakchi and Boye it would be impossible to find the perfect degree between those two goals, which was their reason to use the default hyperparameters.

The last approach to be presented does not employ an LLM to generate QA pairs but instead creates a large corpus of 47,457 medical QA pairs, called *MedQuAD*. It is used by several QA systems presented in Chapter 3 and was published by Abacha and Demner-Fushman [24]. They used a rule-based approach where they crawled data from various trusted medical online sources, taking advantage of the fact that articles from the same source often expose similar patterns. For instance, 116 articles on various cancer types were crawled from the National Cancer Institute (NCI) [1] which reveal a uniform structure across all articles. This uniformity allowed to use the headline of a section to predict the section’s content. For instance, anytime a headline contains the words "may increase the risk of [DISEASE]", the following section mentions the risk groups for the given cancer type. This was used by Abacha and Demner-Fushman to create a QA pair for each of these 116 articles with the question "Who is at risk for [DISEASE]?" and the identified section as an answer.

Although this is a smart approach, it is neither feasible to use the MedQuAD dataset nor to apply a rule-based approach in a similar way. Besides the dataset being in English, this is due to the fact that the answers from each QA pair are sections from the crawled articles. As our agent is voice-based we aim for answers which are one or two sentences long to keep the user interested.

5.1.2. Existing Conversational Agents for Answering Consumer Health Questions

As mentioned in Chapter 4, we retrieved four relevant surveys [1, 8, 21, 34] which we used to find existing CAs through backward search. The study by Welivita and Pu [1] stands out

[1]NCI: `https://www.cancer.gov/types`

as it was recently published and focuses on the consumer health domain.

As we have already presented the survey by Welivita and Pu as well as different conversational agents in Chapter 3, we now aim to present the CA which shares the most similarities with our final approach. After careful consideration, we decided the CHiQA system pubslished by Demner-Fushman et al. [25] is the most similar. Our CA mainly uses a similarity-based approach, which is employed by three other agents: EnquireMe [26], the question-entailment approach by Abacha and Demner-Fushman [28] and CHiQA. All of them were already presented in Chapter 3 hence we will not revisit their implementation details.

Although our agent also emphasizes the conversational aspect, it differs from the enquireMe approach: Our CA does not aim to handle coreferences which is the core concept of enquireMe. Therefore, we thought presenting either the question entailment approach by Abacha and Demner-Fushman [28] or the CHiQA agent [25]. Since the QHiQA system employs the question entailment approach and additionally adds a traditional IR approach, it comes the closest to our solution.

CHiQA is a text-based conversational agent where users can ask one question at a time and retrieve up to five possible answers. The architecture of CHiQA, illustrated in Figure 5.1, consists of two main elements: the traditional IR and the RQE component. Both components generate candidate answers from which the resulting five answers are chosen by a simple team-draft interleaving algorithm [35]. This method ensures a balanced selection from two lists while preserving the original ranking order.

The traditional IR component aims to first extract the type and focus of an incoming question in order to answer the user's question by searching MedlinePlus[2]. The question focus refers to the central disease or syndrome of the question whereas the question type indicates the particular aspect of the disease or syndrome that the question is addressing [36]. For instance, if a user asks "What are the symptoms of diabetes?", the focus would be "diabetes" and the type would be "symptoms". For extracting both, they use a combination of different methods, inlcuding a long short-term memory, i.e. a specific implementation of a neural network, which was trained to identify focus and type of a given question. With the question focus and type extracted, CHiQA then queries a document collection using a variation of the BM25 algorithm to retrieve candidate answers.

Next, we have a detailed look at the RQE approach enabled by CHiQA. The foundation for this approach is the MedQuAD [24] dataset which serves as the knowledge base. The questions from this QA dataset were all indexed as well as synonyms for each question focus and triggers for the question type were added. For a deeper understanding, we provide an example from the RQE approach published by Abacha and Demner-Fushman [28]: Consider the question "What are treatments for Torticollis?" with "Torticollis" being the identified

[2]MedlinePlus: `https://medlineplus.gov/`

focus and "Treatment" being the identified type. To improve searching performance, the following focus synonyms were extracted automatically from the MedQuAD dataset: "Spasmodic torticollis, Wry neck, Loxia, Cervical dystonia". Furthermore, the CHiQA system defines 37 question types and corresponding trigger words. The following triggers were then added to the example question based on the identified type: "relieve, manage, cure, remedy, therapy".

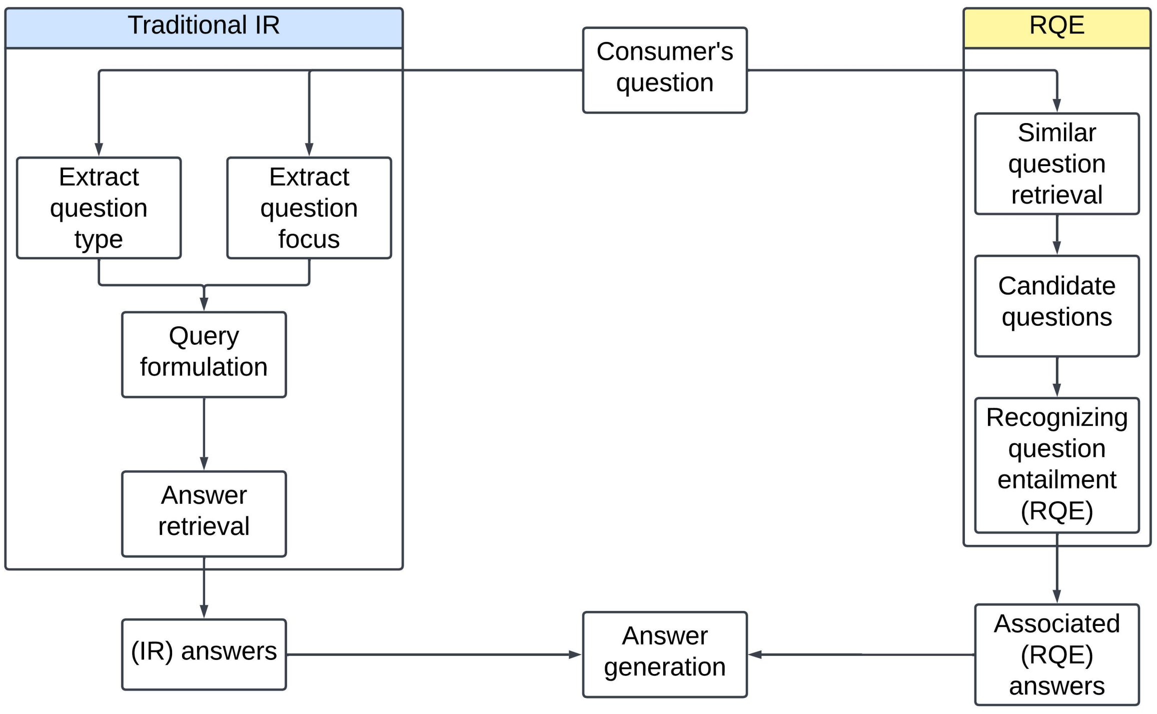


Figure 5.1.: Schematic representation of CHiQA, based on [25]

After all the questions were indexed and enriched with focus synonyms and type triggers, the Terrier search engine[3] was used to retrieve the top 100 questions from the database to a given user question. Next, a logistic regression model is applied to classify entailed questions. As explained in Chapter 3, a question A is entailed by question B if all responses to A are also responses to B. The 100 questions are then ranked by a weighted combination of results from the entailment approach and the similarity score from the Terrier search engine.

In the TREC2017 LiveQA track's medical task, CHiQA achieves an average score of 1.308 which is approximately twice the best average score achieved at the event itself. Demner-Fushman et al. explain this performance with the hybrid approach used by CHiQA [25].

[3]Terrier: `http://terrier.org/`

However, CHiQA was released in 2020 and therefore benefits from three years of research on QA systems.

Besides the insights on how to setup a hybrid approach for QA, the published paper was helpful as it reminded us of the importance of the type of questions a user might ask. This was emphasized during the evaluation of the CHiQA system, which enabled two datasets: The Alexa dataset containing simple questions and the questions from the TREC2017 which often contain complicated nested questions with multiple types and foci. There is a significant drop in performance between those two evaluation methods. This made us more aware of users who first explain a lot of background on their medical history before asking their question, a far more challenging task than answering a question which consists of a single sentence.

5.2. RQ2: Construction of Question-Answering Dataset

In this section we present our pipeline for automatically generating German QA pairs for the following three health-related topics: cardiovascular diseases, mental illnesses, and postural tachycardia syndrome (PoTS). As introduced in Chapter 4 and illustrated in Figure 4.2, this process consists of four main steps: collecting data, sending the data to an LLM to generate the QA pairs, storing the QA pairs with relevant metadata in a database and finally validating the QA pairs.

The main objective of this pipeline is to generate as many and unique QA pairs as possible because they build the foundation for the final CA.

Additionally, after the different steps of the pipeline were presented, we briefly evaluate the quality of our resulting dataset. This is achieved by reviewing the feedback retrieved from the health experts who validated our QA pairs.

5.2.1. Extraction of Data

The first step of our pipeline was to collect data from the sources presented in Chapter 4. In total, from *Apothekenumschau*, we crawled 33 articles about cardiovascular diseases and 14 articles on mental illnesses. From *Nationale VersorgungsLeitlinien*, we obtained one article on blood pressure and 21 on depression. *ANS-Ambulanz der Uniklinik RWTH Aachen* provided one article and one file on PoTS.

Additionally, we extracted data from two articles on blood pressure published by *Deutsche Hochdruckliga e.V. - Deutsche Gesellschaft für Hypertonie und Prävention*. These documents varied in length with most consisting of one or two pages, and some files containing more than 140 pages.

For crawling the data from Apothekenumschau, we used the Python library *Scrapy*[4] which offers a set of functions to simplify the process of extracting HTML source code from

[4]Scrapy: `https://scrapy.org/`

websites. Additionally, we used *Beautiful Soup*[5], another Python library for efficiently processing HTML source code which is known for its robust parsing capabilities. Furthermore, the PyMuPDF library[6] helped with extracting information from PDF files by converting PDF files into a format that could be easily read and processed.

All the data extracted was stored paragraph by paragraph. This would help us when sending the data to ChatGPT for generating the QA pairs mainly for two reasons: First, we cannot send unlimited text to the LLM because there are restrictions on the length of the input text. Second, we noticed, that sending short paragraphs instead of long articles to the LLM would result in more QA pairs.

Additionally, we maintained the hierarchy of our source text by saving the corresponding headlines as well as any subheadings for each paragraph. Since we only sent small paragraphs instead of the whole article or file, those headings can provide relevant context information when prompting the LLM, assuring higher quality QA pairs.

5.2.2. Generation of Question-Answering Pairs

In the second step of our pipeline, we generate the QA pairs using GPT-3.5-Turbo which offers different hyperparameters to configure the model's behavior. However, we decided to retain the default settings since we did not notice any improvements when experimenting with the settings.

In addition, as Kalpakchi and Boye [32] explained, on the one hand we aim to generate creative questions but on the other hand the questions should stay close to the original text. Thus, the default parameters offer a good balance between those two contradicting goals. The only parameter which was configured is the maximum token length to be the highest number possible to avoid any responses being cut off. Our final configurations are therefore as follows:

- temperature of 0.7
- maximum length of 4,000
- "top p" of 0.95
- frequency and presence penalty of 0
- no custom stop sequences

The *temperature* and *top_p* affect the randomness of the model's response. The temperature can range from 0 to 2 and top_p from 0 to 1, where higher values result in less deterministic and more random outputs. The *frequency* and *presence penalty* control the use of creative words by penalizing the repetition of already existing words in the input as well as the text that has been generated so far. The key difference is that the frequency

[5]Beautiful Soup: `https://www.crummy.com/software/BeautifulSoup/`
[6]PyMuPDF: `https://pymupdf.readthedocs.io/en/latest/`

penalty is applied to all words that have been used once whereas the presence penalty is proportional to how often a word has occurred. Both values must be in the range from -2 to 2 with larger values causing more creative results. [37]

We sent our data to an API provided by OpenAI in order to generate the QA pairs. The API request requires multiple parameters: The configurations mentioned above, some secret key and additionally a JavaScript Object Notation (JSON) array, called *message*. The message array can include the following three parts: A *system instruction*, previous chats, and the actual *prompt*. The system instruction influences ChatGPT's behavior for the task. It specifies how to interpret and respond to the given prompt. Moreover, previous chats are provided in the message array because ChatGPT itself is stateless, i.e. it does not store any chat history. Thus, to enable one of ChatGPT's main features, the ability to reference content from previous messages, we need to provide those messages in each prompt. Finally, with the term *prompt* we are exclusively referring to the last message that ChatGPT responds to. While some may use *prompt* to describe the whole message array, we will adhere to our definition.

The message array must follow a specific structure in order to be understood by ChatGPT. It is an array of JSON objects, each object representing a message, and it defines two keys: *role* and *content*. The role specifies who wrote the message and has the possible values *system*, *user*, and *assistant*. If a message indicates the role system, it contains a system instruction, if a message specifies to be written by the user, it was a message sent by a user and in the case of the role being assistant the message was generated by the model. The second key of each message object is the content which is a string value containing the content of the message. The following example illustrates such an array:

```
{
 "role": "sytem",
 "content": <content>
},
{
 "role": "user",
 "content": <content>
},
{
 "role": "assistant",
 "content": <content>
},
{
 "role": "user",
 "content": <content>
},
```

Furthermore, using previous chats supports a method called *few-shot learning*. This is another method for instructing ChatGPT how to handle the prompt. We simply simulate previous chats by writing an example prompt followed by a manually created response that functions as an example output. ChatGPT will mimic the style of the provided examples.

Designing and improving existing prompts is called *prompt engineering*, a challenging task as slight changes can lead to completely new outcomes [38]. In order to achieve satisfying results, we spent many iterations of refining and testing our prompt. In the following, we explain each of the three parts of our message array used to generate our QA pairs.

The first part, the system instruction, can be seen in Listing 5.1. The challenge was to be as concise and short as possible while integrating all our requirements in the system instruction. Mentioning different requirements twice, seemed to result in a shift of which requirement the model puts its focus on but as far as we tested it, reduced the quality of requirements that were mentioned a single time. Therefore, we avoided repeating any requirements.
Besides instructing the system to generate medical QA pairs to a given input context, we integrated the following requirements:

- Avoid hallucinations: The model should only use the information from the input context and not any of its own knowledge.
- Ignore irrelevant information: ChatGPT should ignore information which is not about health topics, e.g. often we extracted paragraphs which give some background information on the author.
- Format response: The resulting QA pairs should be formatted as a JSON which can then be immediately parsed.
- Be short: The model was advised to generate short answers, because our CA is voice-based and users quickly get bored with long responses.
- Allow empty responses: If the model cannot identify any useful text for generating QA pairs, it should return an empty array.
- Optimize quantity: The model should come up with as many QA pairs as possible without violating the previous requirements.

Listing 5.1: System instruction to generate QA pairs

```
{
  "role": "system",
  "content": "Du bist ein Frage-Antwort-Generator für medizinische Fragen in
      verständlicher Sprache. Dir werden ein Thema und Kontext vorgegeben
     und du formulierst ausschließlich aus dem Kontext Fragen-Antwort-Paare,
      die auf wissenschaftlichen Fakten basieren ohne personenbezogene oder
      anekdotische Informationen. Generiere mehrere Frage-Antwort-Paare im
```

```
    JSON Format: [{’question’: ’<frage>’, ’answer’: ’<antwort>’}].
    Generiere sehr kurze Antworten in ein bis maximal zwei Sätzen. Gib []
    zurück, wenn du keine Frage-Antwort-Paar aus dem Kontext formulieren
    kannst. Versuche möglichst viele sinnvolle Frage-Antwort-Paare zu
    generieren."
},
```

Next, we input our examples to employ few-shot learning. We decided to use two examples: One showcasing an input context with useful information where we simulated the system generating as many QA pairs as possible and one input context without any useful information where the system responds with an empty array. The examples formatted as a JSON can be seen here:

Listing 5.2: Few-shot examples

```
{
  "role": "user",
  "content": "Thema: Heuschnupfen Allergie, Kontext: Menschen mit einem
    allergischen Schnupfen müssen häufig niesen, ihre Nase läuft oder ist
    verstopft. Bei starken Beschwerden fühlen sich viele zudem schlapp und
     müde. Wenn auch die Augen tränen und jucken und die Augenlider
    anschwellen, spricht man von einer allergischen Rhinokonjunktivitis -
    einem allergischen Schnupfen, der mit einer Bindehautentzündung
    einhergeht. Die Beschwerden treten nur in den Zeiten des Pollenflugs
    auf."
},
{
  "role": "assistant",
  "content": "[{’question’: ’Welche Symptome treten bei allergischen
    Schnupfen auf?’, ’answer’: ’Symptome von allergischem Schnupfen sind h
    äufiges Niesen, eine laufende oder verstopfte Nase, allgemeine Schwä
    che und Müdigkeit. Zusätzlich können tränende und juckende Augen sowie
     geschwollene Augenlider auftreten.’}]"
},
{
  "role": "assistant",
  "content": "[]"
},
```

The last part of our prompt is the actual input context. Here, we first provide information about the topic that the context is about and then append the text of a single paragraph which we extracted from one of our sources. For this we extracted the headlines to each paragraph as explained in the previous section.

Additionally, most articles have various subheadings which were all extracted in addition to an article’s headline. The prompt always starts with the headline of the overall article and is followed by the content of the paragraph. Since the subheadings often contain relevant information for ChatGPT to better understand the text, they were added to the paragraph. We simply concatenated the paragraph with its heading and any existing subheadings, using newline characters to separate them. The following is one extracted paragraph from an article on "Apothekenumschau" about high blood pressure as the prompt already suggests.

Listing 5.3: Input context

```
{
  "role": "user",
  "content": "Thema: Bluthochdruck: Die schleichende Gefahr, Kontext: Was
     ist normal, ab wann besteht Bluthochdruck?\nWas bedeuten die Blutdruck-
     Werte?\nAuf dem Blutdruck-Messgerät stehen zwei Werte:\n- Der höhere
     Wert ist der systolische Blutdruck. Er entsteht, wenn das Herz sich
     zusammenzieht und das Blut in die Gefäße pumpt - dann ist der Druck am
      höchsten.\n- Der niedrigere Wert ist der diastolische Blutdruck. Er
     entsteht, wenn das Herz sich wieder weitet, um sich erneut mit Blut zu
      füllen.\nBei einem Blutdruck von 120/80 mmHg liegt der systolische
     Druck bei 120, der diastolische bei 80 mmHg."
}
```

The article's headline is introduced by "Thema:" (English: "Topic:") and the paragraph's text by "Kontext:" (English: "Context:"), like the few-shot examples. It is important to stick to the formatting used in any provided few-shot examples to ensure clarity for ChatGPT.

The provided paragraph in this example prompt was extracted from Apothekenumschau[7] from an article called "Bluthochdruck: Die schleichende Gefahr" (English: "High blood pressure: the insidious danger"). The paragraph is preceded by one headline ("Was ist normal, ab wann besteht Bluthochdruck?") and a subheading ("Was bedeuten die Blutdruck-Werte?") which can be spotted at the beginning of the input context provided in the prompt. Moreover, this example illustrates the use of newline characters ("\n") to separate the headline from the subheading as well as the subheading from the paragraph itself.

5.2.3. Further Processing

After generating all our QA pairs, we added some useful metadata and processed the data before storing each pair in a database. In this section, we discuss these processing steps.

Firstly, we ensured that each generated answer was not overly complex since our agent is voice-based and long answers tend to lose listeners' attention. We classified each answer which exceeds a threshold of 25 words as too long. This is since we experienced a lack in attention with answers longer than two normal sized sentences which equals approximately

[7] Apothekenumschau article from example prompt: `https://www.apotheken-umschau.de/krankheiten-symptome/herz-kreislauf-erkrankungen/bluthochdruck-die-schleichende-gefahr-1013191.html`

25 words. The answers which then labelled as too long, were sent to ChatGPT another time to be refactored. To achieve more efficiency, we sent 10 answers formatted as a JSON array to ChatGPT, requesting each to be reformulated in order to be a maximum of 25 words long. In case ChatGPT did not manage to shorten an answer appropriately, we simply removed the corresponding QA pair. Otherwise, the former answer was replaced with the shorter version.

Furthermore, we computed embeddings for each question as these were used by our agent to compare similarities between a user's question and the stored questions. As previously outlined, we used a pre-trained auto-encoding transformer published by Deutsche Telekom AG[8] as this model was fine-tuned on German data. As computing embeddings for a large number of questions costs time, it should be done in advance to using our agent.

With the embeddings added to each QA pair, we could perform another processing step: We checked whether each question generated was semantically unique within our collection. Therefore, we computed the cosine similarity of one question to all the other questions. We assumed two questions to be semantically identical if the exceeded a similarity of 96%. This threshold was chosen to be very strict as it avoids removing any closely related questions which might achieve high similarity but still are unique. This step helped cleaning our resulting dataset as we had different articles covering the same topic which sometimes led to QA pairs asking the same question with slightly different sentence structure.

The next processing step was to add some information about which text was used to generate the QA pair. This included adding the context that was provided when prompting ChatGPT as well as adding the name of the source this context came from.

Lastly, we prepared our dataset for recommending new questions after the user's initial question was successfully answered. This step involved the following two challenges: Identifying questions that generally arouse the user's interest and suggesting questions related to the user's original query.

Our ideas to address these challenges will be presented in Section 5.3.3 where the recommendation system of our final agent is explained in detail. For now, we aim to mention all the information that we store with each QA pair. Thus, for identifying interesting questions we labelled each question that is less than eight words long as what we call a *beginner question*. These labelled questions were then rephrased using ChatGPT in order to be used as engaging follow-up questions. We instructed ChatGPT rewrite a given question to begin with "Do you want to know ...". For instance, the question "What is hypertension?" would result in "Do you want to know what hypertension is?".

Additionally, to ensure recommending a related question, we assigned each QA pair to a subject. The resulting subject were then organized into a three-level taxonomy. The top level consisted of the three main topics that we crawled data on, i.e. cardiovascular

[8]Model for Embeddings: https://huggingface.co/deutsche-telekom/gbert-large-paraphrase-cosine

diseases, mental illnesses and PoTS. The middle level divided each of these topics into what we called *subtopics*, while the last level contained all the subjects which we initially assigned our QA pairs to, we called these *subsubtopics*. Further details follow in Section 5.3.3.

To sum all of our processing steps up, we present the columns of the final QA dataset with brief explanations:

- *question*: generated question
- *answer*: generated answer
- *complexity_level*: labeling a question as beginner or intermediate question
- *recommendation*: containing a rephrased version of each beginner question
- *context*: context used to generate the QA pair
- *source*: source information in human readable format
- *source_ref*: raw source information, e.g. link to a website
- *topic*: top level of taxonomy
- *subtopic*: middle level of taxonomy
- *subsubtopic*: bottom level of taxonomy

Overall, after removing duplicate questions in this processing step, we generated a total of 3,581 QA pairs. After a first review by some health experts, we received the feedback that the answers which we shortened on purpose are sometimes too short to convey the relevant information to answer the corresponding question. Therefore, we send all our crawled paragraphs a second time to ChatGPT and added all the QA pairs which were longer than 25 words. These were labelled as complex questions which we are currently ignoring with our agent. Possibly a future version of our agent could use them if a user seems highly interested and the system thinks answer which are slightly longer will not reduce the user's attention. This leads to a total of 4,335 QA pairs in our final dataset.

Most of these pairs covers the topic *depression* with 858 questions and the second most QA pairs were created on the topic *high blood pressure* with a number of 737 pairs. Furthermore, we labelled 1,823 QA pairs as beginner questions and 754 as complex questions.

5.2.4. Validation

The final step of our pipeline QA generation pipeline is the validation of each QA pair by a team of health experts. This step is crucial to avoid any misleading or false information which could lead to serious consequences in the context of health-related questions. Therefore, all our QA pairs were sent to different health professionals where each is specialized on one of the three domains covered by our QA pairs. They received an Excel-sheet containing the questions, corresponding answers and assigned topics from our taxonomy. The task for the professionals was then to provide feedback on correctness of the answer, correctness of the assigned topic and text quality for each pair.

The health experts are not completely finished by the time this thesis is submitted but provided feedback for most of the QA pairs. We present this feedback in order to assess the quality of our generated dataset and determine potential limitations of the QA generation pipeline.

The majority of the QA pairs were approved by the health professionals. However, the feedback also highlighted different issues in the generated dataset which we have divided into the following three categories: answers containing wrong information, answers which are wrong because they miss important information and useless but correct QA pairs.

The first category was rare but still some QA pairs contained incorrect information. For instance, the answer to "Was ist das Posturale Tachykardiesyndrom?" (English: "What is the postural tachycardia syndrom?") contained the information that PoTS is accompanied by a drop in blood pressure. Our health experts pointed this mistake out, as the blood pressure eventually increases but not necessarily drops when suffering from PoTS. Interestingly, ChatGPT generated another QA pair where the question was semantically identical, but the provided answer did not include any wrong information. When further investigating the provided context used for generating the faulty answer, it was noticeable that the context did not contain this false information. This suggests that the error was a result of the model hallucinating and not following the instruction to only use provided knowledge.
Furthermore, the input context did not contain any useful. Thus, this was the only QA pair being generated for this context. It seems as if ChatGPT struggles to answer with an empty array which would have been the expected answer and rather generates some faulty data. This was observed in a few other cases.

The second category of errors were answers which omitted important information. They appeared more frequently than the first category but still were relatively rare. The following is an example from our constructed dataset: The question "Für welche Art von Depression empfehlen Fachleute Johanniskraut?" (English: "What type of depression do experts recommend St. John's wort for?") was answered with "Fachleute empfehlen Johanniskraut nur bei einer leichten oder mittelschweren Depression." (English: "Experts recommend St. John's wort only for mild or moderate depression."). Although the answer is correct, it misses one relevant detail: St. John's wort should be bought from a pharmacy and prescribed by a doctor to ensure correct dosage. Even though this information was included in the input context sent to ChatGPT, it was not included in the response. In the case of this example, it is most likely because the LLM did not recognizes this information as crucial. However, we have found other cases where it seems as if shortening all the answers to 25 words led to removing important information. As the health experts noted this effect as well, we decided to create the 754 questions with complex answers mentioned previously.

The final category of issues includes QA pairs which are useless while still being correct. These pairs do not add any faulty information but pollute the dataset with superfluous information. An example for this category would be the question "Bei wie vielen Personen zeigt die Behandlung mit Medikamenten nach 12 Wochen Erfolg?" (English: "In how many people is the treatment with medication successful after 12 weeks?"). It was generated based on a text discussing the efficacy of medication for depression but the LLM never mentioned the word depression in the question or even in the answer. This makes it almost impossible to match it to a related user question, as the question in isolation could refer to any disease.

In addition, this category contains multiple cases where ChatGPT refers to individuals, mainly the authors of the crawled articles, who ChatGPT was actually instructed to ignore.

Besides these three categories, the health experts also annotated some QA pairs to be using overly complex language. This leads to answers being not accessible to the user.

In conclusion, the number of errors was relatively low but with a total of 4,300 QA pairs, it was expected to generate some faulty data. The received feedback confirmed the quality of our dataset and underpinned the importance of validating each QA pair instead of blindly relying on ChatGPT's results.

5.3. RQ3: Development of a Conversational Agent

In this section we explain the implementation details of our resulting CA including the matching of QA pairs, traditional IR, disambiguation and our recommendation-system. We have developed a pipeline to process an incoming question which employs a similarity-based and an information retrieval approach. We begin this section by giving a brief overview of our architecture, next describe our two approaches on QA, then continue the main ideas behind our recommendation-system and finally describe the setup of our agent in Dialogflow.

5.3.1. Architecture of a Conversational Agent

The architecture of the resulting agent can be seen in Figure 5.2. The agent consists of three main components: A question analysis component, a response generation component, and a recommendation component. The first two are explained in more detail in Section 5.3.2 and the recommendation component in Section 5.3.3.

When the user initiates a conversation by asking a question, the agent starts with preprocessing the input. First it will be classified as either health-related or -unrelated. If the question is health-unrelated, the agent triggers a web search using the Bing API[9] provided by Microsoft.

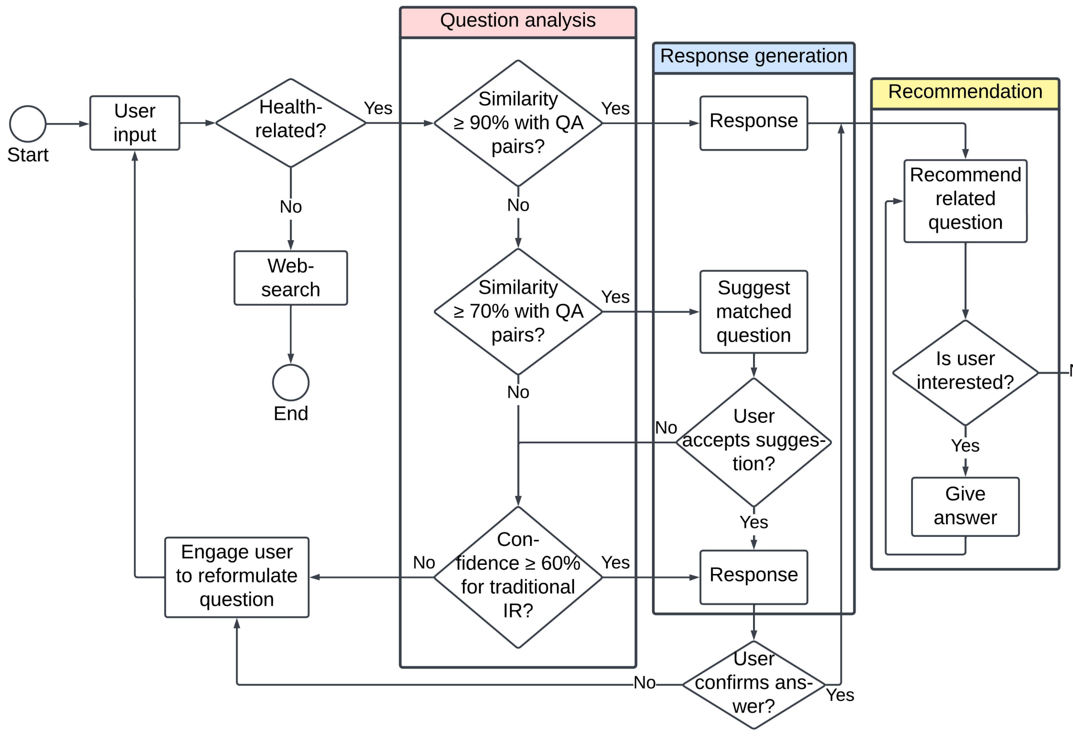


Figure 5.2.: Flowchart showing conversation steps with conversational agent

Instead, if the question is health-related, the agent performs another preprocessing step, which is not mentioned in Figure 5.2 for simplicity reasons: It checks whether the topic of the incoming question is covered by any of the QA pairs in our dataset. This is simply done, by verifying that there is any question with a similarity score of at least 40%.

If the question is health-related but the topic is not covered by any of our QA pairs, the agent politely informs the user about the specific areas it can provide answers to. Otherwise, the flow of events continues with retrieving the most similar question among the collection of QA pairs. If a question with a similarity score of 90% or greater exists, we call this a *direct match*. The corresponding answer will be returned, and the agent continues with the recommendation component. If the most similar question has got a similarity score greater than or equal to 70% and less than 90%, the system will perform a disambiguation step by asking the user if the question is what the user was looking for. If

[9]Bing API: https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

the user responds with a "No", the system checks whether there is another question with a similarity score in the range of 70% to 90% to perform the same disambiguation step a second time.

If there is no such question or the user responds with "No" again, the third and last step of the question analysis component is performed: Through traditional IR the CA tries to find a passage in a document that contains an answer to the user's query, more details in Section 5.3.2. If no such document can be found, the system engages the user to reformulate the question. If either the disambiguation steps were successful or the traditional IR approach could find a passage in a document that answers the user's question with a confidence score of 60% or more, the response will be output. In contrast to the scenario of a direct match, in this case the agent will ask the user whether the provided answer was satisfying. If not, the user is engaged to try again, similar to when the traditional IR approach fails. However, if the CA manages to correctly answer the response, the agent continues with the recommendation component.

The recommendation component simply suggests new questions which were labelled as *beginner questions* during the construction of our dataset. The system continues recommending new questions until the user does not show interest anymore, again more details in Section 5.3.3.

5.3.2. Question Analysis Component

The agent employs a pipeline of two main components in order to answer an incoming question: The first component is based on a question-similarity approach and the second component uses traditional IR. The former is the agent's main focus on answering a given question while the latter can be considered a fallback strategy.

As previously outlined, the foundation of the agent's QA skill is based on a question-similarity approach, using the *German BERT large paraphrase cosine*[10] model published by *Deutsche Telekom AG* on HuggingFace. It is an auto-encoding transformer that was fine-tuned on German input data. We use it to map our questions to a 1,024-dimensional vector space, as explained in Chapter 2. The embedding for the question of each QA pair is precomputed and stored in the database. As a database, we use Milvus[11] which offers a convenient platform as it can be run through Python script which can be included into our existing Flask application. Additionally, it is highly efficient in computing vector operations. As our dataset contains roughly 4,300 QA pairs, and we compare an incoming question to each of our embedded questions, efficiently computing the similarity is crucial to our application. As pointed out in Chapter 2, we decided to use cosine similarity, as it only requires a single operation, the dot product, if we normalize all vectors in advance. These

[10]BERT-model for embeddings: `https://huggingface.co/deutsche-telekom/gbert-large-paraphrase-cosine`

[11]Milvus: `https://milvus.io/`

design decisions avoid any interruptions in the flow of conversation by quickly responding to a user's question. Furthermore, if the knowledge base of our agent should be expanded in the future, retrieval of similar questions does not become a bottleneck that easily.

As the user asks a question, the system embeds the question and computes the cosine similarity of the input question and all the stored questions. The cosine similarity will result in a value in the range of -1 to 1. In this context, we often write about similarity in percentage as it is more intuitive. If we e.g. write about a similarity score of 90%, this would mean that the cosine similarity returned the value 0.9. After all similarity scores were computed, the agent retrieves the question with the highest score, the *best match*. We then compare the corresponding similarity score to two different thresholds to find out if the best match is sufficiently similar to be a direct match or otherwise to be used for disambiguation.

Deciding what threshold to apply, e.g. to determine a direct match, is a challenging task. A softer threshold, e.g. 80% instead of 90% leads to more false positives, whereas a stricter threshold results in more false negatives. False positives in this context refer to questions that have a different semantic meaning than the user's prompt but are interpreted as identical. Vice versa, false negatives are questions that are labelled as not similar enough to the user's query although they have the same semantic meaning. As the perfect threshold highly depends on many settings such as the use-case and the model used, it is crucial to experiment with different values. Additionally, considering the idea behind the system which is developed can help by identifying a suitable threshold: In the health domain it is important to avoid any false information, which is the main reason for us to tend in the direction of a stricter threshold, avoiding false answers. Additionally, the disambiguation step in our pipeline allows to reduce the number of false positives.

After experimenting with different values, our final architecture applies a threshold of 90% to identify a direct match. If the best match fulfills this threshold, the system responds with the corresponding answer of the matched QA pair and continues with the recommendation component explained in Section 5.3.3. Otherwise, the agent checks whether a disambiguation step can be performed. Based on the same thoughts as discussed for the previous threshold, we came up with a threshold of 70% which needs to be fulfilled for performing the disambiguation step. If the similarity score is in the range of 70-90%, the agent asks whether the user maybe wanted to ask the question from the best match. If the user denies this suggestion, the system checks if the second-best match also achieves a similarity score greater than 70%. If that is the case, the disambiguation step is repeated for the second question. If the user decides to accept either the first or the second suggested question, the agent returns the corresponding answer. The answer is followed by what we call a *confirmation step*, that is a question to confirm that the user's question was correctly answered. This helps with avoiding false positives, as the concept of disambiguation and applying a softer threshold automatically leads to an increase in false positives. If the user then confirms the answer, the agent continues with the recommendation component and otherwise the user is engaged to reformulate the question.

Any questions during the disambiguation step that were denied by the user will be added to the *blacklist*, a list of QA pairs that the user was not interested in. This list helps with avoiding suggesting a question twice to the user. If the user is engaged to rephrase a question and the system performs a disambiguation step with the rephrased question, it does not suggest a question from any QA pair on the blacklist.

Furthermore, if either the best match could not achieve a similarity score of 70% or the questions suggested for disambiguation were denied by the user, a fallback method is performed using traditional IR. As outlined in Chapter 2, traditional IR mainly differs from similarity-based QA in the format of the information that is stored and used to answer a user's question. Thus, we set up a second database that contains raw documents about various health topics. As we have already collected different articles for generating our QA pairs, we reused the same data for populating the database which is then used for traditional IR. In order to determine a response that is around three sentences long, we again setup a small pipeline of multiple steps as illustrated in Figure 5.3.

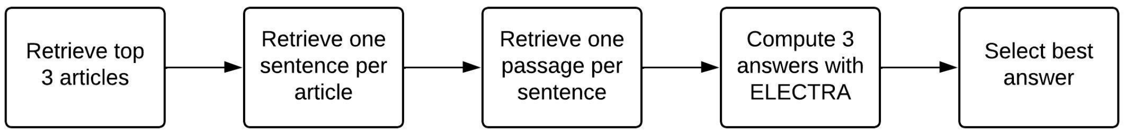

Retrieve top 3 articles
Retrieve one sentence per article
Retrieve one passage per sentence
Compute 3 answers with ELECTRA
Select best answer

Figure 5.3.: Pipeline to answer a question with traditional IR

The first step of the traditional IR pipeline selects three candidate articles using BM25. We decided to use the Python library *rank-bm25*[12] which provides multiple implementations of BM25 all presented in a paper published by Trotman et al. [39]. We decided to use the basic BM25 implementation as the results were satisfying.

We plan on using a transformer model to extract the answer from our article, which can take up to a minute to process an entire article. Because we aim to respond in less than five seconds, we performed the next two steps from our pipeline which select a small passage from each of the three candidate articles. This drastically reduces the transformer's computing time. Firstly, we generate embeddings for every sentence within each article of our database, using the same transformer as in the similarity-based approach. We then select the best matching sentence for each of the three candidate articles, by comparing the embedded question asked by the user with every sentence in one article. We repeat this step for all three articles which results in a selection of three sentences, each from a different article.

[12]rank-bm25: `https://pypi.org/project/rank-bm25/`

In the next step of our pipeline, we retrieve a passage per article based on the matched sentences. We simply choose the 300 characters that precede and the 300 characters that follow our selected sentence in each article. This results in one text string consisting of three parts: 300 characters before the matched sentence, the matched sentence and 300 characters after the matched sentence. If the sentence should be at the end or start of an article which makes it impossible to either select 300 preceding or following characters, we add more characters on the other end of the passage to still append 600 characters to our selected sentence.

After we have retrieved one passage per article, we use a transformer model to extract a response from each passage. We again decided to use a model provided by Deutsche Telekom AG on HuggingFace[13] which is designed for extracting answers given a question and a text. The model uses the ELECTRA architecture and was fine-tuned on a German QA dataset. For each of our passages the model returns a short answer, usually just a few words within that passage and a confidence score indicating the level of confidence of an answer being correct. Among those three answers, we select the answer with the highest confidence score and apply a threshold of 60%, as any lower confidence score seems to include incorrect answers.

If the best answer fulfills the threshold, we add the previous and the following sentence to the answer extracted by ELECTRA and return this as the system's response. Afterwards the system's confirmation step is executed. If the confidence score is below 60%, the user is engaged to reformulate the question.

As previously explained, the similarity-based approach is the foundation of our solution whereas the traditional IR component can be seen as a fallback method. If the similarity-based approach fails, we try to find an answer using traditional IR. As we have already crawled the documents used within our traditional IR component for generating the QA pairs, we decided to implement our traditional IR pipeline.

5.3.3. Recommendation Component

After a user's query was correctly answered, the agent engages the user in a conversation by recommending new topics, illustrated as the recommendation component in Figure 5.2. The idea is to tell the user a question similar to the previously answered question and ask whether the user is interested in knowing the answer to that question. If the user shows interest the answer is returned and the system recommends another question, again asking whether the user is interested in getting to know the answer to the new question. This process is repeated until the user expresses not being interested in any suggested question.

The main challenge for this component lies in developing a mechanism to choose a

[13]QA model: `https://huggingface.co/deutsche-telekom/electra-base-de-squad2`

question that might be interesting to the user, as the goal is to keep the user engaged to teach them interesting new facts. Each conversation starts with the user asking the agent a question. Only after the agent successfully responded, the recommendation component becomes active. We assume that the user is most likely interested in the topic of the initial question which started the conversation and therefore, recommend related questions.

In order to know the topic of a single question, we decided to create a three-level taxonomy that assigns each QA pair to what we call a *subsubtopic* and each subsubtopic is then assigned to a *subtopic* which again is assigned to a *topic*. Therefore, the subsubtopics represent the bottom layer, the leaf-nodes, and the topics represent the top layer of the taxonomy. Furthermore, the subsubtopics represent each name of the articles that were crawled to construct the dataset whilst. When crawling the PDF files, we manually assigned them to a subsubtopic, so each generated QA pair could be linked to a subsubtopic. The topics in our taxonomy were the three fields covered by our source documents: cardiovascular diseases, mental illnesses and PoTS.

Next, we added subtopics to differentiate our topics in a more granular way and at the same time group similar subsubtopics together in one category. For cardiovascular diseases we used the categories presented by *Berufsverband Deutscher Internistinnen und Internisten*[14] as a first draft. We further refined the presented categories in communication with a physician. For the subtopics on mental illnesses, we created the taxonomy together with some health experts. Additionally, for PoTS, the last topic covered by our QA pairs, we decided to not create a taxonomy, as we only generated roughly 100 QA pairs on this topic which do not need to be further categorized.

Figure 5.4 shows the resulting taxonomy for the topic mental illnesses to illustrate the three levels that the taxonomy employs. Furthermore, the taxonomy for cardiovascular diseases can be found in the appendix A.1. However, both taxonomies are designed in German, as the source documents and the agent itself use German.

After the taxonomy was successfully created and approved by a team of health experts, we could assign each QA pair to a subtopic and topic as the subsubtopic was already assigned when generating the QA pairs based on the source document. The same could be done to the articles and documents that we stored in order to perform traditional IR.

The taxonomy allows the agent to suggest a question based on the subtopic the user showed interest in. The subtopic is simply determined by either extracting it from the QA pair which was used to answer the user's question or if the traditional IR was performed, we extract the subtopic from the article used to create the system's response.

Our QA pairs served as a pool of questions which can be potentially recommended to the user. However, we cannot simply recommend any question to the user that covers the same subtopic, as our QA pairs also consist of very specific or complicated questions. In

[14] Subtopics for cardiovascular diseases: `https://www.internisten-im-netz.de/fachgebiete/herz-kreislauf/herz-kreislauf-erkrankungen.html`

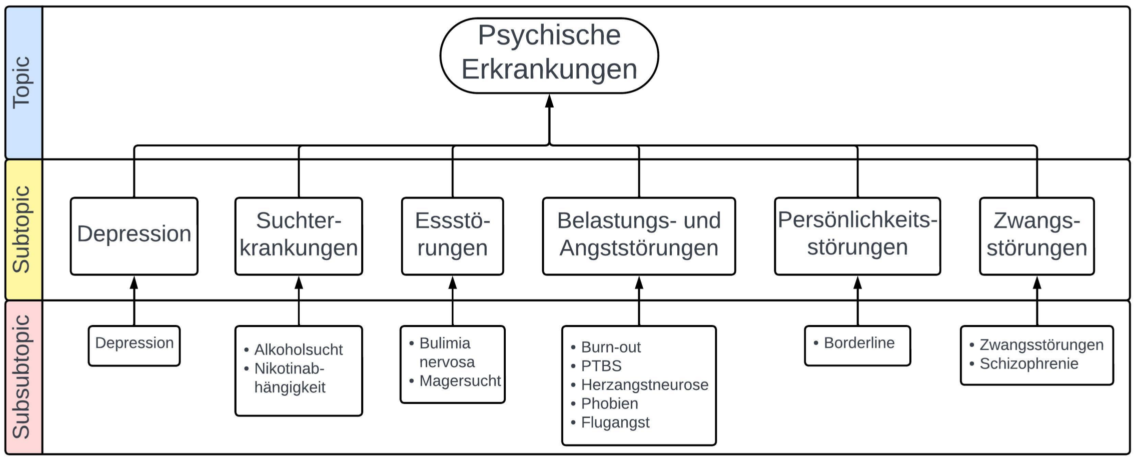


Figure 5.4.: Taxonomy for mental illnesses used by recommendation component

order to engage the user in a conversation it seems reasonable to recommend what we labelled a *beginner question*, i.e. a question which is understood by anyone and applicable to most users. For instance, the question "What can I do about high blood pressure if I have diabetes?" would only be interesting to a specific group of users who have diabetes because of the conditional sentence. As we noticed that short questions often follow the pattern "What is ...?" or "What are the causes of ...?", we labelled each QA pair which is less than eight words long as suitable for recommendations.

This simple approach achieved promising results, as it labelled long and complicated questions as intermediate. The only downside of this method is that it leads to labelling questions which could be suitable beginner questions as intermediate. This is due to the fact that eight words is a very strict threshold. For instance, the question "Welche Symptome können bei zu geringer Salzkonzentration im Blut auftreten?" (English: "What symptoms can occur if the salt concentration in the blood is too low?") is a question that could catch the user's interest because it is simple and a topic which affects many people. As this question exceeds the threshold of seven words, this is an example from our QA pairs which was labelled as intermediate.

Since with a dataset consisting of 4,300 QA pairs, we can afford to miss a few possible beginner questions, this easy-to-implement approach highly improves the quality of recommended questions. In total 1,245 questions were labelled as beginner questions with the fewest beginner questions from the subtopic "Persönlichkeitsstörungen" (English: "Personality disorders"), that is five questions. However, for the two subtopics which are covered by the most of our source documents, i.e. "Belastungs- und Angststörungen" (English: "Stress and anxiety disorders") and "Bluthochdruck und Blutdruckstörungen" (English: "High blood pressure and blood pressure disorders"), we labelled 252 and 229 respectively as beginner questions.

Our final recommendation system then recommends new questions to the user based on the identified subtopic using the questions labelled as beginner questions. The agent remembers which questions have already been answered so that they are not recommended again. This includes both the question of the QA pair that was used to answer the user's question, as well as questions that were answered in the course of the recommendations. If a user is continuously accepting the suggested questions, it is possible that the agent runs out of recommendation questions. In that case, the agent randomly chooses another subtopic from within the same topic and continues recommending questions only from that subtopic until it runs out of questions again.

5.3.4. Dialogflow

The previous sections introduced the algorithms and techniques performed at different steps of the flowchart. For allowing a smooth conversation which can map the user's utterances to specific scenarios in our flowchart, we used Dialogflow and its intents. In the following, we will discuss the intents implemented by our Dialogflow agent. An overview of all existing intents is given in Table 5.1. We will start with presenting all the intents which are not used for answering a health question and then finish with the intents that are necessary for our QA pipeline.

The first intent is called *help* and simply should give a short introduction on how to interact with our agent. The next intent, *web.search*, is triggered whenever a question seems to not be related to any health topic. It can be either triggered if the agent identifies the incoming prompt as an open-domain question, e.g. "Who is Barack Obama?", or via a Dialogflow event, i.e. a method to trigger another intent from within the process of handling an initially matched intent.

The intent that will be matched whenever the user asks a new health-related question, is the *health.search* intent. It is trained with questions on different health-related topics. The linked webhook service employs our question-analysis component and decides whether the request can be solved through a direct match, a disambiguation step, traditional IR or cannot be solved at all.

The *health.disambiguation*, *health.confirmation* and *health.recommendation* intents are triggered by a user's "yes" or "no" and depend on so called *input contexts*. Input contexts are a feature provided by Dialogflow to correctly assign any user utterance to an intent. In this case they are important as these three intents defined the same trigger words. Each of these three intents is used whenever the system previously asked a question and expects a "yes" or "no" as an answer from the user. For *health.disambiguation*, the system suggested a disambiguation question which the user can accept or deny, for *health.confirmation* the system wants to confirm whether it answered the user's question correctly and for *health.recommendation* the system asked whether the user is interested in a specific question which also needs to be answered with a "yes" or a "no".

Therefore, if one of these questions was asked by the agent, the system internally sets a context which uniquely identifies one of the three scenarios. Each of the three intents then requires its unique context to be triggered which avoids any ambiguities whenever a user responds with a "yes" or a "no" as the previously set context determines which intent needs to be triggered. For instance, if the *health.search* intent retrieved a direct match and thus, aims to continue with the recommendation component, it would set a recommendation-context. Additionally, it would output the recommended question after answering the initial question. The user receives a suggested question, and the recommendation-context is set. Any "yes" or "no" will then trigger the *health.recommendation* intent.

Furthermore, the contexts are used to add relevant information for the flow of the conversation. In the example with the recommendation-context, this could be the index of the previously answered question to avoid recommending this question again. Additionally, the lifespans of the contexts used for this purpose are always set to one which makes them disappear after the next input provided by the user. This is important as the three intents use the same trigger words and thus could interfere with each other if multiple of these input contexts are set.

The intent *health.traditional* is only triggered via events either when the system is handling a new question and could not find a suitable QA pair or if the disambiguation step was unsuccessful. If the system found an answer through traditional IR, it responds with the retrieved passage and asks for confirmation including setting the *confirmation* context. Otherwise, it engages the user to reformulate the question. If that is the case, in the background all previously suggested questions are preserved in a context called *prev_conv* to avoid suggesting them again. For example, if the user asks a question and the system could not find a direct match, it might find two questions suitable for our disambiguation step. If the user denies the first disambiguation question and afterwards the second one, the system tries to use traditional IR. If traditional IR was unsuccessful, it creates the *prev_conv*-context to store the indices of both questions which were denied by the user. If the user now reformulates the question, our agent will not suggest the same questions again, as the user was not interested in them. The same context is set if the user responds with a "no" in the confirmation step.

Furthermore, the agent consists of the *health.source* intent which can be triggered anytime that the system returned a response. It provides information about the source that was used to either generate the corresponding QA pair or if traditional IR was performed, the source article of the extracted passage. The process of retrieving the source information is straightforward as it is linked to each QA pair and article in our databases. Therefore, whenever the agent retrieves a QA pair or an article to answer a question, it extracts the source information and stores it in a context that is called *source*. This context is required to be set in order to trigger the *health.source* intent.

Table 5.1.: Dialogflow agent intents and their corresponding user inputs and agent responses

Intent name	**Example input**	**Agent response**
help	How does this work?	Provides a short introduction on how to interact with the agent.
web.search	Who is Barack Obama?	Triggers when an open-domain question is asked or via an event; provides an answer not related to health.
health.search	What are the symptoms of flu?	Matches health-related questions and suggests a recommendation-question or performs IR.
health.recommendation	Yes, I'm interested. No, I'm not interested.	Provides the corresponding answer to the recommended question or acknowledges disinterest.
health.disambiguation	Yes, I'm interested. No, I'm not interested.	Provides the corresponding answer to the disambiguation question or continues with traditional IR.
health.confirmation	Yes, my question was answered. No, my question was not answered.	Asks for confirmation if the agent's response answered the user's question.
health.traditional	(Triggered via events)	Provides an answer from traditional IR or asks the user to reformulate the question.
health.source	Where did this information come from?	Provides source information for the QA pair or the article used in the response.

5.4. RQ4: Evaluation of CA

As outlined in Chapter 4, the evaluation is divided into an automatic evaluation and test user evaluation. In this section we present results from both methods, addressing the research question *"What evaluation methods can be used to assess the performance and effectiveness of the developed system?"*.

5.4.1. Automatic Evaluation

For the automatic evaluation we randomly selected 200 QA pairs from our dataset and prompted GPT-3.5-Turbo to reformulate our selected questions. As we are sending a request to the API provided by OpenAI, we need to define one message array per prompt, as outlined in Section 5.2.2.

The first message to ChatGPT contains a short system instruction to reformulate any incoming questions. Additionally, we provide two few-shot examples, each example consisting of one question sent by the user and a simulated response from the system which rephrases the question accordingly. The following is an example of what a final message sent to ChatGPT looked like:

```
{
   "role": "system",
   "content": "Du erhälst eine Frage und du musst die Frage umformulieren.",

},
{
   "role": "user",
   "content": "Was kann ein Grund für einen zu niedrigen Blutdruck bei ä
      lteren Menschen sein?",
},
{
   "role": "assistant",
   "content": "Warum könnten ältere Menschen einen niedrigen Blutdruck
      haben?",
},
{
   "role": "user",
   "content": "Was ist der FAST-Test?"
},
{
   "role": "assistant",
   "content": "Wie lässt sich der FAST-Test beschreiben?",
},
{
   "role": "user",
   "content": "Wie stellt der Arzt eine KHK fest?"
}
```

At the end of this array, the prompt is given which simply consists of the question that we aim to rephrase. This message is what we consider the *basic prompt*, as the model is not instructed to rewrite the question in a specific style. However, we adjusted the hyperparameters aiming to encourage more creative formulations. The following parameters were chosen:

- temperature of 2

- maximum length of 4,000
- "top p" of 1
- frequency and presence penalty of 2
- no custom stop sequences

As mentioned previously, the temperature and top p impact the randomness for the model's output with higher values leading to more random outputs [37]. Large values for frequency and presence penalty engage the model to use more creative words, penalizing the reuse of words already seen. As we aim to assess the efficacy of our similarity-based approach, we decided to maximize all those four values. This leads to the most random and creative reformulations achievable with GPT-3.5-Turbo.

The second sample of 200 questions was then send to ChatGPT with the instruction to rewrite the questions from the first-person perspective of an elderly person. Similar to the basic prompt, we added a few-shot example and at the end of the message we submitted the question to be rewritten. The following is an example of the final message sent to ChatGPT:

```
{
   "role": "system",
   "content": "Du redest wie ein alter Mensch. Deine Aufgabe ist, es eine
      gegebene Frage aus der Ich-Perspektive zu wiederholen.",
},
{
   "role": "user",
   "content": "Was kann ein Grund für einen zu niedrigen Blutdruck bei ä
      lteren Menschen sein?",
},
{
   "role": "assistant",
   "content": "Woran kann es liegen, dass ich als älterer Mensch einen zu
      niedrigen Blutdruck habe?",
},
{
   "role": "user",
   "content": "Wie unterscheiden sich primäre und sekundäre Krampfadern?"
}
```

The aim with the second sample was to test the matching performance for the target user group for the ALPHA smartwatch which mainly is used by elderly people. Furthermore, we noticed that all the stored QA pairs are using an impersonal perspective. As some users might ask questions from their own perspective, we were interested in evaluating how well our agent can match these questions and thus instructed the system to use the first-person perspective. Additionally, with the first prompt we were interested in generating creative reformulations. With this prompt, we focus on the system instruction and therefore we used the default settings for the hyperparameters:

- temperature of 0.7
- maximum length of 4,000
- "top p" of 0.95
- frequency and presence penalty of 0
- no custom stop sequences

For our evaluation we identified five scenarios that could occur when matching a reformulated question with one of our QA pairs. To avoid confusion, we will refer to the original question as $Q_{original}$, the rephrased question as $Q_{rephrased}$, and the question matched by the agent with Q_{match}.

The three first scenarios are considered correct matches, characterized by $Q_{rephrased}$ and Q_{match} being identical. The first scenario requires $Q_{rephrased}$ to be at least 90% similar to Q_{match}, previously defined as a direct match. The second scenario involves a similarity score between $Q_{rephrased}$ and Q_{match} of less than 90% but more than 70%, considered a disambiguation match. The third scenario, which never occurred in our results but is mentioned for completeness, involves $Q_{rephrased}$ and Q_{match} being the same but not meeting the disambiguation threshold.

The last two scenarios are what we defined as incorrect matches, meaning $Q_{original}$ and Q_{match} are not the same. The fourth scenario occurs if Q_{match} is semantically the same as $Q_{original}$, indicating that even though the match was incorrect, the question would still be answered correctly. The fifth scenario occurs if Q_{match} is different in meaning from $Q_{original}$, marking a clear mistake.

To classify whether an incorrect match falls under the fourth of fifth scenario, we reviewed each incorrect match. We were strict in our evaluation, only considering questions semantically the same if individual words were replaced with synonyms or the sentence structure was slightly modified.

The results of the automatic evaluation can be seen in Table 5.2 and will be further discussed in the following.

For the basic prompt, the distribution of match types indicates that most questions (70%) resulted in direct matches, where the original and rephrased questions achieved at least 90% similarity. Disambiguation matches accounted for 15% of the outcomes. Semantically correct matches, representing cases where the matched question did not exactly match the original but was semantically equivalent, comprised 12%. Incorrect matches, where the matched question diverged in meaning from the original, were the least common, constituting only 4%.

	correct match		**incorrect match**	
	direct match	disambiguation match	semantically correct match	incorrect match
basic prompt	139 / 0.70	30 / 0.15	23 / 0.12	8 / 0.04
old person prompt	137 / 0.69	43 / 0.22	18 / 0.09	2 / 0.01

Table 5.2.: Automatic performance evaluation of the conversational agent: Distribution and frequency of match types based on similarity score for a random sample of 200 questions

For the old person prompt, the outcomes are even more favorable: In total 91% of the questions were correctly matched compared to 85% for the basic prompt. However, the old person prompt achieved less direct matches but 7% more disambiguation matches. Similarly, the old person prompt resulted in fewer incorrect matches compared to the basic prompt, with only 1% of the questions being matched to a question that differs in meaning from the original, as opposed to 4%.

The results indicate that the agent is proficient in recognizing similar questions, but there are noteworthy observations regarding the incorrect matches. For instance, in some cases the incorrectly matched question is still closely related to the original question that was reformulated. For example, we found the following incorrect match within our 200 samples from the basic prompt: The original question was "Welche kleinen Aktivitäten können bei Depressionen helfen?" (English: "What small activities can help with depression?") and the matched question was "Welche Möglichkeiten gibt es zur Behandlung von Depressionen?" (English: "What options are there for treating depression?"). In this case, the answer provided could still be relevant to the user's query, as the agent might include small activities among the treatment options for depression.

Moreover, the rephrasing process for one question, influenced by the selection of hyperparameters, was overly creative. The question "Welches Blutfett steigt bei der familiären Hypercholesterinämie an?" (English: "Which blood fat rises in familial hypercholesterolemia?") was changed to "Bei welcher Erkrankung steigt das Blutfett aufgrund von Genmutationen an?" (English: "In which disease does blood fat increase due to genetic mutations?") which removed the name of the disease which the question is interested in. This rephrasing makes accurate matching challenging, yet the error might not necessarily lie with the agent.

These findings indicate that the incorrect matches are not as problematic as they first appear. However, there are also some incorrect matches which highlight challenges of the agent's matching performance, for example the handling of abbreviations. The collection of QA pairs is very inconsistent with the use of abbreviations, depending on the disease being covered. While abbreviations are employed for certain diseases, others that could also be abbreviated are not. Moreover, within the dataset, there are instances where one QA pair for a specific disease might use the abbreviation, whereas another pair for the same disease does not.

This inconsistency poses difficulties for accurate question matching as shown by one example from the incorrect match of the old person prompt, illustrated in Figure 5.5. The first question is the original question from our QA pairs collection, it is about coronary artery disease (CHD) and uses the abbreviation. In contrast, the second question which was generated by ChatGPT uses the long version. Thus, the agent incorrectly matched it to a question not using the abbreviation, as seen in the third question. Despite the semantic similarity of the original and matched questions, this instance was classified as a mismatch because we only allowed minimal variations to count an incorrect match as a semantically correct match. Nevertheless, this example effectively addresses a weakness of our similarity-based approach.

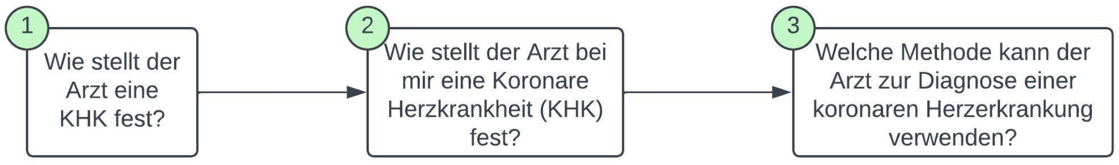

1
Wie stellt der Arzt eine KHK fest?
2
Wie stellt der Arzt bei mir eine Koronare Herzkrankheit (KHK) fest?
3
Welche Methode kann der Arzt zur Diagnose einer koronaren Herzerkrankung verwenden?

Figure 5.5.: Example for a question being randomly selected from the collection of QA pairs (1), reformulated by ChatGPT (2) and incorrectly matched to another question from the collection (3)

Furthermore, the old person prompt achieved 7% more disambiguation matches than the basic prompt. As we instructed ChatGPT to rewrite the questions from the first-person perspective, we assume this lowered the similarity score for matched question and led to less frequently meeting the threshold for the direct match. Regardless of whether this assumption, further investigation showed that the first-person perspective has a slight but relevant impact on the overall similarity score. Therefore, this could be addressed in future improvements by preprocessing any input questions to remove any words related to first-person perspective.

Additionally, 12% of the questions using the basic prompt and 9% of the questions using the old person prompt were matched to a semantically correct question. As a reminder, this means that the agent's matched question only shows slight deviations compared to the original question, such as individual words that have been exchanged or a different sentence structure.

Therefore, these results indicate the occurrences of duplicate questions in the constructed dataset. While having a large amount of QA pairs increases the chances of matching a user's question, our goal is to minimize redundancy in the dataset as the following example should illustrate:

Consider a user asking an ambiguous question, leading to the agent performing a disambiguation step. Further assume that at least two questions meet the disambiguation threshold: the most similar question does not reflect what the user is interested but the second most similar question does. As the disambiguation step presents both, the user's question can be answered. In a dataset polluted with duplicate questions, there is a risk that the irrelevant question is contained in multiple times in the dataset just phrased differently. This could lead to the two most similar questions being different variants of the irrelevant question. As only the two most similar questions will be suggested, the user would not see the relevant question. This scenario could negatively impact the user experience as the system could not provide a response.

5.4.2. Test User Evaluation

The second part of the evaluation was a qualitative feedback from a group of 10 test users who interacted with our CA for the first time. Therefore, we created a survey which instructed the participants to ask our agent any questions related to the three topics cardiovascular diseases, mental illnesses, and PoTS. After this brief introduction, they had five minutes to interact with the agent without any further details while we were observing their conversation with the agent. When the five minutes were over, we provided a more detailed instruction about the agent's functionalities. This often included explaining different features that the user has not noticed yet or an explanation of the type of questions our agent is able to answer, e.g. its inability to handle coreferences.

Afterwards, the user should answer questions capturing the user experience, system's usability, the relevance and accuracy of generated responses and the agent's effectiveness in engaging the user to learn more. At the end of the survey, we collected demographic information, including age, technical knowledge, and familiarity with medical topics. The list of questions is provided in the appendix A.1.

Based on the answers, we knew that the majority of the participants had little knowledge on medical topics but considered themselves as intermediate in technical topics. The average age of the test group was 26.1 years.
Furthermore, it is important to note that the users interacted with a chatbot version of our agent instead of a voice-interface. This was due to the fact that by the time when the participants evaluated the agent, the voice-based version had not been deployed yet.

The overall feedback was positive as shown by the answers to the initial question asking how the users would rate their experience. However, most answers also mentioned difficulties when interacting with our agent, which we will now discuss in detail.

The first five minutes of each test already provided deeper insights in possible expectations and misunderstandings users might experience when interacting with our agent for the first time. As we did not fully explain the capabilities and limitations of our agent, we could analyze the user's expectations through observing the participants' questions.

The most common misunderstanding was that users interacted with our agent as if it was an LLM, such as ChatGPT, likely because it is the most-used CA. This was noted whenever users asked a question which coreferenced previous messages. Additionally, in some cases users responded with "no" to a disambiguation question asked by the agent and within the same message asked a new question to the agent. This did not result in the outcome the user expected, as the agent would only register the "no" response and ignore the new question. This is due to the agent matching the "no"-entity to the *health.disambiguation* intent which engages the user to rephrase the question. However, to answer any question, the *health.search* intent must be triggered.

Another common misunderstanding was the assumption that the agent could function like a doctor. This means participants explained some fictional symptoms and asked the agent for a diagnose. As the agent is not capable of diagnosing any symptoms, the users noticed after a few questions that this is not possible.

Discovering any misunderstandings or ambiguities was helpful as it underscores the importance of communicating the agent's capabilities to future users of the ALPHA smartwatch.

Besides these misunderstandings about the agent's functionality, we now present different scenarios and questions where the agent did not perform as expected.

Moving on to one of these scenarios, we found that some users were unsure what to begin asking questions about. Therefore, they asked questions on higher-level topics, such as "Which cardiovascular diseases are there?". This was challenging for the agent, as the dataset of QA pairs contained questions on specific diseases rather than the higher-level topic "cardiovascular diseases". This is due to none of the crawled articles used to generate the QA pairs covered the topic "cardiovascular diseases". To address this problem, a solution could be to either manually add a few QA pairs about the top-level topics from our taxonomy or finding a suitable article covering these topics to generate new QA pairs on these topics.

Furthermore, in one case the agent suggested a wrong question in the disambiguation step, even after the user was rephrasing the input. We investigated the reason for this problem and discovered that the database contained a QA pair which would have answered the question, but it was worded strangely. After further research, we realized the source article used the same wording and therefore, ChatGPT must have copied it. Although the strangely phrased question met the threshold to be selected for a disambiguation step, the database contained nearly 10 other questions that were more similar. Thus, they were suggested instead.

This problem could be solved by implementing the feedback provided in the validation step of the QA generation pipeline, which we have not retrieved by the time of the evaluation. Additionally, it is important to highlight that this incident occurred only once. Usually, if a question could not be answered correctly, this was due to no QA pair existing in our dataset that covered the topic.

Besides challenges concerning the matching performance, we have noticed different unexpected behaviors by the participants. The first thing that we noticed was no user responded with "no" to the disambiguation or confirmation question asked by our agent. If users were not satisfied with any suggested question, they immediately rephrased their question instead of responding with "no". This behavior led to the agent never suggesting a second disambiguation question. Also the feature designed to avoid suggesting disambiguation questions a second time after a user reformulates their question could never be explored, as no disambiguation questions were denied and thus not added to the blacklist.

Another observation related to unexpected user behavior was that some participants did not engage with any questions asked by the agent. This behavior however might be cause by the setup of our evaluation, as we used a text-based instead of a voice-based agent. In a chatbot environment, it is easier to overlook or ignore any information that seems irrelevant. Conversely, in a voice-based output, the information that is best remembered usually is the last sentence from the response which would be the question asked by our agent. Thus, we would need to further investigate the interaction of users with a voice-based version of our agent.

Furthermore, we found that the traditional IR component of our agent never answered any question. Mostly, this was due to the similarity-based approach finding a response first. However, there were cases where the traditional IR component was triggered but failed to return a suitable response. Further investigation showed that the traditional IR component struggled to retrieve the article which contains the answer to a given question. Therefore, the three retrieved articles did not contain the responses which made it impossible for the transformer model to extract a correct answer from a given passage. This model calculates a confidence score to assess how well the extracted response answers the input question. Since the agent requires the confidence score to be at least 60% and none of the extracted answers met this threshold, the system never returned any answer.

Since the traditional IR approach was only a fallback method, the overall performance and user experience did not suffer from these failures. However, in the future this component could be further improved.

The feedback helped us to identify some limitations and challenges of our agent. Despite the issues mentioned, once the participants understood the concept behind our agent, the experiences were mostly positive. Additionally, nine out of 10 participants stated in the survey that they appreciated the recommendation questions.

However, it is important to acknowledge that the participants did not align perfectly with the target group of the ALPHA smartwatch, which is mostly used by elderly and thus less technical skilled users. For instance, it is unlikely that older individuals expect this agent to be like ChatGPT.

In conclusion, the agent reveals some limitations which could be solved once the feedback from the health experts is integrated into the dataset. Despite these challenges, the feedback was generally positive, indicating a promising direction for future enhancements. These positive results underscore the potential of the agent to bridge the gap between medical professional language and consumer language by employing CAs.

However, further evaluations need to be performed, especially with a group of older people which might expose different challenges when interacting with our agent.

6. Discussion

This thesis focuses on exploring the design and implementation of a CA that answers consumer questions to health-related topics. This objective was answered in two steps. Firstly, generating a dataset of QA pairs that was validated by a group of health professionals. Secondly, the CA was developed, enabling a pipeline of multiple steps to provide suitable answers to any incoming inquiries. Afterwards, the agent was evaluated employing two different strategies. In this chapter, we discuss the feedback from the health professionals and the results from both evaluation strategies to critically assess our approach.

6.1. Key Findings

The constructed dataset builds the foundation of our final agent, as high quality QA pairs result in more accurate matching. Based on the feedback provided by the health professionals, the dataset generally shows a high quality apart from a few minor errors. We noticed that the quality heavily relies on the provided input context. In many cases, errors in the data were traced back to low-quality input contexts provided to ChatGPT. As the model could not identify any useful information, it simply hallucinated. Additionally, sometimes the input context provided the necessary details to create a correct answer, but the model ignored relevant information. This occurred either because the model failed to identify what was relevant or because we instructed the LLM to shorten an answer that exceeded our 25 words limit.

It's important to note, however, that such errors were rare and mainly confirmed our decision to perform a validation step in our pipeline of creating QA pairs instead of relying solely on an LLM. Thus, we could remove or correct any of the misleading information.

Overall, the similarity-based approach of our CA was efficient and accurate, as both evaluation strategies confirmed. The matching performance in our automatic evaluation exceeded expectations: Only 4% of the questions using a basic prompt and 1% using the old person prompt could not be answered correctly. This finding proved that the embeddings computed by our transformer could effectively capture the semantic meaning of questions. Additionally, the decision to include a disambiguation step in our QA pipeline was also beneficial: For the old person prompt, the agent successfully matched 22% of all questions by performing a disambiguation step. Without this feature, the agent would have instructed the user to rephrase their question and lowering the user experience drastically.

Furthermore, when evaluating with the test users, the agent could answer the majority of questions if two conditions were met: The user asked a question on a topic that was covered, and the question was of a general nature and e.g., not asking for a diagnose with a set of given symptoms.

Besides the agent being able to answer most questions, the recommendation component successfully engaged the participants of the evaluation in a conversation. In our survey, nine out of 10 users reported positive experiences with the recommended questions. Although the approach was fairly simple, it seemed to be effective. The combination of selecting *beginner questions* by the length of the sentence and utilizing a small taxonomy to suggest related questions created a seamless conversation.

6.2. Challenges

Despite the agent's success in answering health-related questions and generally satisfying user experiences, it faces different challenges and limitations which we will highlight in this section.

One issue is that the dataset contains some QA pairs with very similar semantic meaning. Having many unnecessary QA pairs in our database makes it more likely that an irrelevant question matches with the input question. Moreover, many similar QA pairs reduce the effectiveness of the disambiguation step. For instance, if a user asks an ambiguous question, the agent could retrieve two matches: the first match is incorrect but the second would answer the user's question. The disambiguation step would suggest both and thus, could answer the ambiguous question. However, if the question from the incorrect match is contained in the dataset multiple times but slightly different phrased, the disambiguation step might rank the correct question lower and not present it to the user. Additionally, considering that the dataset might grow and include millions of QA pairs, it is not feasible to store a single question multiple times but differently phrased.

This highlights a specific trade-off that we encountered: On the one hand, we aim to generate a vast amount of QA pairs as this increases the chances of finding a match for a given question. On the other hand, we avoid generating too many QA pairs as this enhances the probability of semantically identical questions leading to the previously outlined difficulties.

Some duplicate QA pairs could have been caused by using multiple articles covering the same topic to generate the dataset. As outlined when explaining our QA pair generation pipeline, the pipeline has already removed duplicate questions. However, the similarity-threshold was quite strict because we preferred having duplicate QA pairs over removing unique pairs. To improve the dataset's quality, it might be useful to choose a softer threshold. Alternatively, the system could start with a strict threshold and lower it for subsequent articles covering topics that were already used. This strategy might be a compromise between keeping unique questions while removing duplicates.

Besides limitations concerning the constructed dataset, the matching performance of the agent faced some difficulties. The results show that the perspective used when asking a question had an impact on the similarity score. When a user asked a question using the impersonal view, it received a higher similarity score compared to the same question asked from a first-person perspective. This is due to most of the QA pairs being written impersonally. Although this effect does not significantly impact the similarity score, it could reduce the overall user experience, as for instance the matched question more frequently triggers the disambiguation step instead of a direct match.

This could be improved by adding a preprocessing step before performing the similarity-search. This step would involve removing any occurrences of words related to the first-person perspective from the user's question, such as "I" or "me".

Another challenge regarding the matching performance involves the use of acronyms. For instance, if all QA pairs mention a specific disease by its full name, while the user's question contains its acronym, it becomes almost impossible to correctly match the question. Addressing this issue could also involve a preprocessing step that replaces any occurrences of acronyms in the user's question with their full form. However, this approach might become overly complex as the dataset expands and the system needs to maintain an exhaustive list of acronyms.

Alternatively, the dataset could contain two versions for each QA pair that contains a disease that also has a short from: The written out and the abbreviated version. This ensures that no matter which version the user is using, there is a QA pair to match it with. Yet, also this approach has its downsides as it might lead to pollution of the dataset.

Furthermore, although the similarity-based approach led to satisfying results, the traditional IR component was rather ineffective. During the evaluation by the test users, this component has never returned any response because the confidence score of the extracted responses did not reach the required threshold. Further analysis revealed that this component mainly struggled to identify the article that contains the answer to a question, rather than in extracting the answer from a correctly retrieved article. As the presented solutions from literature showed, it might be beneficial to extract the focus and type of a question to retrieve candidate articles rather than the entire question.

However, as traditional IR was intended to be a fallback method and most of the cases the similarity-based approach was able to find an answer, the system's overall performance was not impacted. Still, in the future it could be advantageous to implement a hybrid solution, making the agent more robust.

Another critical aspect to discuss is that the evaluation was not fully representative for the target group of the ALPHA smartwatch. The test users were on average 26.1 years old and mostly state to have a technical background, while the main users of ALPHA are elderly people and thus, they are less familiar with technology. Moreover, the evaluation was limited to just 10 test users, which is insufficient to draw meaningful conclusions on the agent. However, the evaluation intended to gather initial feedback which can be used to determine the most relevant challenges exposed by the agent.

7. Conclusion

7.1. Summary

This thesis addressed the design and implementation of a voice-based CA for responding to CHQs. Therefore, the thesis focused on two main pillars: Firstly, the construction of a dataset which could then be used by the second pillar, the development of the voice-based agent. The main objective was to improve accessibility to health information, especially for people less familiar with technology.

The process of answering this central question was guided by four research questions, starting with analyzing existing methods on answering CHQs. This review identified three main approaches: traditional IR, knowledge graphs and entailment-/similarity-based approaches. Additionally, when answering CHQs, literature showed that one main challenge is the gap between medical jargon and consumer language. A similarity-based approach was chosen for its ability to bridge this gap: The numerous QA pairs stored simply employ a consumer-friendly language.

The decision to use a similarity-based approach required constructing a suitable dataset as a lack of high quality QA datasets in the medical domain was noted. Therefore, addressing the second research question, a dataset of QA pairs was constructed through a pipeline of four central steps: Collecting data from multiple sources, sending the collected data to GPT-3.5-Turbo to generate the QA pairs, adding useful metadata to each pair and validating the final dataset. The validation was performed by different health experts who evaluated the text quality as well as the correctness of each QA pair.

The third research question focused on the development of a CA that employs a QA pipeline to respond to a given question. This pipeline includes several components: a preprocessing component, a similarity-based approach, a disambiguation step, a traditional IR approach and a recommendation component to engage the user to learn more.

After the dataset was created and the agent developed, the final research question addressed the evaluation of the system. The evaluation was performed in two steps: First focusing on the matching performance and second using a group of test users to assess the conversational features. The findings showed that the agent was effective at answering CHQs as stated by the test users. Furthermore, the agent successfully handled ambiguities and engaged users to learn new topics.

In conclusion, the results of this thesis demonstrate that leveraging an LLM for constructing a dataset of QA pairs is a sufficient approach. Moreover, this dataset could be used to develop an agent that successfully answers CHQs, underscoring the value of CAs in the consumer health domain to improve accessibility of health information.

7.2. Future Work

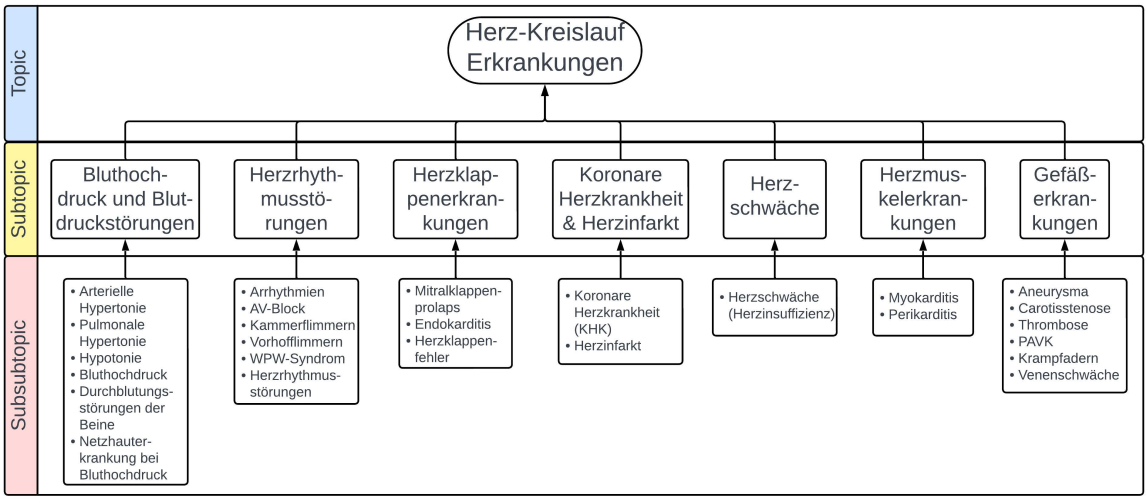
As this thesis has contributed a new method for creating an agent in the consumer health domain, several new questions and possibilities arise for future work. First of all, the agent's hybrid approach could be further enhanced by improving the traditional IR component. Rather than working as a fallback, this method could work collaboratively with the similarity-based component, where the agent always considers both approaches when creating a response.

Furthermore, future work could involve the handling of coreferences throughout the conversation with a user. This involves maintaining awareness of the topic being discussed, even if it is not directly mentioned in follow-up questions. Since many of the test users already anticipated this behavior, implementing it could improve the overall user experience.

Investigating more representative evaluation methods presents another interesting direction. For instance, the TREC2017 dataset for evaluating English consumer health questions could be translated into German to evaluate the agent's performance and to establish a benchmark for comparing the performance of German-speaking medical agents. Furthermore, an evaluation with a larger and more diverse group of test users would lead to more conclusive results.

Additionally, adding a component to the system that remembers questions which were not successfully matched could give valuable insights. This would not only be interesting for the analysis of weaknesses of the agent but also these unmatched questions could be used for adding new topics to the agent's knowledge base. This feature could iteratively expand the agent's coverage of health-related topics.

A. Appendix



Topic
Subtopic
Subsubtopic
Herz-Kreislauf Erkrankungen
Bluthoch-druck und Blut-druckstörungen
Herzrhyth-musstö-rungen
Herzklap-penerkran-kungen
Koronare Herzkrankheit & Herzinfarkt
Herz-schwäche
Herzmus-kelerkran-kungen
Gefäß-erkran-kungen
• Arterielle Hypertonie • Pulmonale Hypertonie • Hypotonie • Bluthochdruck • Durchblutungs-störungen der Beine • Netzhauter-krankung bei Bluthochdruck
• Arrhythmien • AV-Block • Kammerflimmern • Vorhofflimmern • WPW-Syndrom • Herzrhythmus-störungen
• Mitralklappen-prolaps • Endokarditis • Herzklappen-fehler
• Koronare Herzkrankheit (KHK) • Herzinfarkt
• Herzschwäche (Herzinsuffizienz)
• Myokarditis • Perikarditis
• Aneurysma • Carotisstenose • Thrombose • PAVK • Krampfadern • Venenschwäche

Figure A.1.: Taxonomy for cardiovascular diseases used by recommendation component

Type of Question	**Question**
User Experience and Satisfaction	How would you rate your overall experience with the agent?
	What did you like most when interacting with the agent?
	What did you like least when interacting with the agent?
	Would you use the agent again? If yes, in which scenarios?
Usability and Accessibility	How easy was it to interact with the agent?
	How clear and understandable were the agent's responses?
Relevance and Accuracy	How relevant were the answers to your questions?
	How accurate were the answers provided by the agent?
	How relevant were the questions recommended by the agent?
Engagement and Curiosity	Did the recommended questions motivate you to explore more?
	How engaging did you find the conversation with the agent?
Personal Information	How old are you?
	How would you rate your knowledge on medical topics?
	How would you rate your knowledge on technical topics?

Table A.1.: Survey Questions

List of Figures

List of Tables

Acronyms

Bibliography

[1] A. Welivita and P. Pu. “A survey of consumer health question answering systems”. In: *AI Magazine* 44.4 (2023), pp. 482–507. DOI: https://doi.org/10.1002/aaai.12140. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aaai.12140.

[2] P. Zweigenbaum. “Question answering in biomedicine”. In: *Proceedings Workshop on Natural Language Processing for Question Answering* 2005 (2005), pp. 1–4.

[3] V. Nguyen. “Question Answering in the Biomedical Domain”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Ed. by F. Alva-Manchego, E. Choi, and D. Khashabi. Florence, Italy: Association for Computational Linguistics, 2019, pp. 54–63. DOI: 10.18653/v1/P19-2008. URL: https://aclanthology.org/P19-2008.

[4] M. Allouch, A. Azaria, and R. Azoulay. “Conversational Agents: Goals, Technologies, Vision and Challenges”. In: *Sensors* 21.24 (2021). ISSN: 1424-8220. DOI: 10.3390/s21248448. URL: https://www.mdpi.com/1424-8220/21/24/8448.

[5] B. Bui Huu Trung. *Multimodal Dialogue Management - State of the art*. Undefined. CTIT Technical Report Series 06-01. Imported from CTIT. Netherlands: Centre for Telematics and Information Technology (CTIT), Jan. 2006.

[6] H. Weld, X. Huang, S. Long, J. Poon, and S. C. Han. “A Survey of Joint Intent Detection and Slot Filling Models in Natural Language Understanding”. In: *ACM Comput. Surv.* 55.8 (Dec. 2022). ISSN: 0360-0300. DOI: 10.1145/3547138. URL: https://doi.org/10.1145/3547138.

[7] I. Dagkoulis and L. Moussiades. “A Comparative Evaluation of Chatbot Development Platforms”. In: *Proceedings of the 26th Pan-Hellenic Conference on Informatics*. PCI ’22. , Athens, Greece, Association for Computing Machinery, 2023, pp. 322–328. ISBN: 9781450398541. DOI: 10.1145/3575879.3576012. URL: https://doi.org/10.1145/3575879.3576012.

[8] J. L. Z. Montenegro, C. A. da Costa, and R. da Rosa Righi. “Survey of conversational agents in health”. In: *Expert Systems with Applications* 129 (2019), pp. 56–67. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.03.054. URL: https://www.sciencedirect.com/science/article/pii/S0957417419302283.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[10] H. Face. *The Hugging Face Course, 2022*. https://huggingface.co/course. [Online; accessed 02.02.2024]. 2022.

[11] K. Ethayarajh. “How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings”. In: *CoRR* abs/1909.00512 (2019). arXiv: 1909.00512. URL: http://arxiv.org/abs/1909.00512.

[12] K. Clark, M. Luong, Q. V. Le, and C. D. Manning. “ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators”. In: *CoRR* abs/2003.10555 (2020). arXiv: 2003.10555. URL: https://arxiv.org/abs/2003.10555.

[13] Y. Bengio, R. Ducharme, and P. Vincent. “A Neural Probabilistic Language Model”. In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

[14] J. Gómez and P.-P. Vázquez. “An Empirical Evaluation of Document Embeddings and Similarity Metrics for Scientific Articles”. In: *Applied Sciences* 12.11 (2022). ISSN: 2076-3417. DOI: 10.3390/app12115664. URL: https://www.mdpi.com/2076-3417/12/11/5664.

[15] A. Singhal. “Modern Information Retrieval: A Brief Overview”. In: *IEEE Data Eng. Bull.* 24 (2001), pp. 35–43.

[16] T. Vrbanec and A. Meštrović. “Comparison study of unsupervised paraphrase detection: Deep learning—The key for semantic similarity detection”. In: *Expert Systems* 40.9 (2023), e13386. DOI: https://doi.org/10.1111/exsy.13386. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/exsy.13386. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13386.

[17] V. U. Thompson, C. Panchev, and M. Oakes. “Performance evaluation of similarity measures on similar and dissimilar text retrieval”. In: *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*. Vol. 01. 2015, pp. 577–584.

[18] S. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. “Okapi at TREC-3”. In: Jan. 1994, pp. 109–126.

[19] R. Seitz. In: (Mar. 2020). URL: https://kmwllc.com/index.php/2020/03/20/understanding-tf-idf-and-bm-25/.

[20] G. Amati. “BM25”. In: *Encyclopedia of Database Systems*. Ed. by L. LIU and M. T. ÖZSU. Boston, MA: Springer US, 2009, pp. 257–260. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_921. URL: https://doi.org/10.1007/978-0-387-39940-9_921.

[21] M. A. Calijorne Soares and F. S. Parreiras. “A literature review on question answering techniques, paradigms and systems”. In: *Journal of King Saud University - Computer and Information Sciences* 32.6 (2020), pp. 635–646. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2018.08.005. URL: https://www.sciencedirect.com/science/article/pii/S131915781830082X.

[22] D. Wang and E. Nyberg. “CMU OAQA at TREC 2017 LiveQA: A Neural Dual Entailment Approach for Question Paraphrase Identification”. In: *Proceedings of The Twenty-Sixth Text REtrieval Conference (TREC)* (2017). URL: https://trec.nist.gov/pubs/trec26/papers/CMU-OAQA-QA.pdf.

[23] Y. Yang, J. Yu, Y. Hu, X. Xu, and E. Nyberg. “CMU LiveMedQA at TREC 2017 LiveQA: A Consumer Health Question Answering System”. In: *CoRR* abs/1711.05789 (2017). URL: http://arxiv.org/abs/1711.05789.

[24] A. B. Abacha and D. Demner-Fushman. “A Question-Entailment Approach to Question Answering”. In: *CoRR* abs/1901.08079 (2019). arXiv: 1901.08079. URL: http://arxiv.org/abs/1901.08079.

[25] D. Demner-Fushman, Y. Mrabet, and A. Ben Abacha. “Consumer health information and question answering: helping consumers find answers to their health-related information needs”. In: *Journal of the American Medical Informatics Association* 27.2 (Oct. 2019), pp. 194–201. ISSN: 1527-974X. DOI: 10.1093/jamia/ocz152. eprint: https://academic.oup.com/jamia/article-pdf/27/2/194/34152452/ocz152.pdf. URL: https://doi.org/10.1093/jamia/ocz152.

[26] W. Wong, J. Thangarajah, and L. Padgham. “Contextual question answering for the health domain”. In: *Journal of the American Society for Information Science and Technology* 63.11 (2012), pp. 2313–2327. DOI: https://doi.org/10.1002/asi.22733. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.22733.

[27] Y. P. Asma Ben Abacha Eugene Agichtein and D. Demner-Fushman. “Overview of the Medical Question Answering Task at Trec 2017 Liveqa”. In: *Proceedings of The Twenty-Sixth Text REtrieval Conference (TREC)* (2017). URL: https://trec.nist.gov/pubs/trec26/papers/Overview-QA.pdf.

[28] A. B. Abacha and D. Demner-Fushman. “A question-entailment approach to question answering”. In: *BMC Bioinformatics* 20.511 (2019). URL: https://doi.org/10.1186/s12859-019-3119-4.

[29] M.-D. Olvera-Lobo and J. Gutiérrez-Artacho. “Open- vs. Restricted-Domain QA Systems in the Biomedical Field”. In: *Journal of Information Science* 37.2 (2011), pp. 152–162. DOI: 10.1177/0165551511398575. eprint: https://doi.org/10.1177/0165551511398575. URL: https://doi.org/10.1177/0165551511398575.

[30] Z. Gu, Q. Wang, F. Li, and Y. Ou. “Design of Intelligent QA for Self-learning of College Students Based on BERT”. In: *ISCTT 2021; 6th International Conference on Information Science, Computer Technology and Transportation*. 2021, pp. 1–5.

[31] A. Lamurias, D. Sousa, and F. M. Couto. “Generating Biomedical Question Answering Corpora From Q&A Forums”. In: *IEEE Access* 8 (2020), pp. 161042–161051. DOI: `10.1109/ACCESS.2020.3020868`.

[32] D. Kalpakchi and J. Boye. “Quasi: a synthetic Question-Answering dataset in Swedish using GPT-3 and zero-shot learning”. In: *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*. Ed. by T. Alumäe and M. Fishel. Tórshavn, Faroe Islands: University of Tartu Library, May 2023, pp. 477–491. URL: `https://aclanthology.org/2023.nodalida-1.48`.

[33] V. Samuel, H. Aynaou, A. G. Chowdhury, K. V. Ramanan, and A. Chadha. *Can LLMs Augment Low-Resource Reading Comprehension Datasets? Opportunities and Challenges*. 2023. arXiv: `2309.12426 [cs.CL]`.

[34] E. Mutabazi, J. Ni, G. Tang, and W. Cao. “A Review on Medical Textual Question Answering Systems Based on Deep Learning Approaches”. In: *Applied Sciences* 11.12 (2021). ISSN: 2076-3417. DOI: `10.3390/app11125456`. URL: `https://www.mdpi.com/2076-3417/11/12/5456`.

[35] A. Chuklin, A. Schuth, K. Zhou, and M. D. Rijke. “A Comparative Analysis of Interleaving Methods for Aggregated Search”. In: *ACM Trans. Inf. Syst.* 33.2 (Feb. 2015). ISSN: 1046-8188. DOI: `10.1145/2668120`. URL: `https://doi.org/10.1145/2668120`.

[36] A. Deardorff, K. Masterton, K. Roberts, H. Kilicoglu, and D. Demner-Fushman. “A protocol-driven approach to automatically finding authoritative answers to consumer health questions in online resources”. In: *Journal of the Association for Information Science and Technology* 68.7 (2017), pp. 1724–1736. DOI: `https://doi.org/10.1002/asi.23806`. eprint: `https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/asi.23806`. URL: `https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.23806`.

[37] *OpenAI API Reference: Chat Create*. Accessed: 2024-02-09. URL: `https://platform.openai.com/docs/api-reference/chat/create`.

[38] B. Meskó. “Prompt Engineering as an Important Emerging Skill for Medical Professionals: Tutorial”. In: *J Med Internet Res* 25 (Oct. 2023), e50638. ISSN: 1438-8871. DOI: `10.2196/50638`. URL: `http://www.ncbi.nlm.nih.gov/pubmed/37792434`.

[39] A. Trotman, A. Puurula, and B. Burgess. “Improvements to BM25 and Language Models Examined”. In: ADCS ’14. Melbourne, VIC, Australia: Association for Computing Machinery, 2014, pp. 58–65. ISBN: 9781450330008. DOI: `10.1145/2682862.2682863`. URL: `https://doi.org/10.1145/2682862.2682863`.